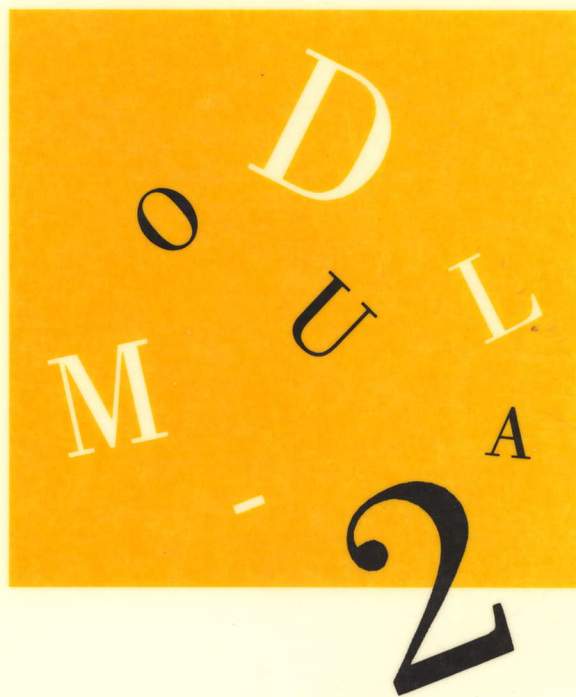


Benchmark Modula-2



A V A N T
G A R D E
S O F T W A R E

Benchmark Modula-2

Benchmark Modula-2 is the programming environment everyone has been waiting for. Here are just a few of its amazing features:

- Extremely easy to use interactive programming environment featuring an integrated editor, compiler, linker and loader.
- Perform complex editing tasks at turbo speed with the EMACS based editor which has over 150 commands. The editor supports multiple files and windows.
- Instantaneous compilation, just press [F2] and the current file is compiled at an amazing rate of 10,000 lines/minute with bursts of up to 30,000 lines/minute.
- Fix errors right on the spot, just press [F1] and the cursor is moved to the position of the next error in the source file.
- Instantaneous linking, just press [F3] and your program is linked at lightning speed. Most programs can be linked in under 5 seconds.
- Instant execution, just press [F4] and your program begins to execute. Execution speed is incredibly fast and the executable files are very compact.

THE EXPERTS USE BENCHMARK MODULA-2

"If you were waiting for an Amiga version of Turbo-PASCAL, forget it! Get the Benchmark package and learn Modula-2. You'll never want to program in Pascal again."

-Richie Bielak, Amazing Computing Magazine

"Frankly, I've been spoiled by the well-integrated Benchmark environment, so anything else pales in comparison."

-Steve Faiwizewski, Amazing Computing Magazine

"The novice programmer will find Benchmark Modula-2 a joy to use and the professional programmer can use Benchmark Modula-2 to squeeze the last ounce of performance out of the Amiga."

-Martin Murray, author of PowerWindows, INOVATRONICS Inc.

"The compiler can be kept in memory (to save loading and reloading) and the libraries, linker and editor fit on one disk. I have been able to happily run the compiler on an Amiga A500 with a single drive — quite a feat, especially with hardly any disk-swapping! The development 'environment' brings back memories of Turbo-Pascal on the PC . . ."

-DJ, Enigma Magazine (U.K.)

THE MODULA-2 LANGUAGE

Modula-2 is the successor language to Pascal and is specifically designed for development rather than just classroom instruction. Modula-2 is very easy to learn because its syntax is the clearest of any of the modern high-level languages. Programmers familiar with Pascal, "C" or another high-level language will be able to start using Modula-2 immediately. The syntax of Modula-2 is so clear that a program may be easily understood even if the original author did not include comments. Universities are beginning to use Modula-2 as a replacement for Pascal because it is simple enough for introductory programming courses, yet powerful enough to solve complex problems.

FEATURES

- A text editor based on EMACS.
- A Modula-2 compiler.
- A Modula-2 linker.
- Access to all Amiga functions: AmigaDOS, Exec, Graphics, Intuition, Layers . . .
- Standard modules: FileSystem, Terminal, InOut, Storage, MathLib0 . . .
- 700 pages of professionally written documentation.
- A large collection of demonstration programs.

PERFORMANCE

Program	Source (lines)	Source (files)	Compile (seconds)	Link (seconds)	Run (seconds)	Size (bytes)
Sieve	50	1	0.5	0.3	4.8	1230
Othello	1320	11	60	2.5	N/A	15602
Scimpi	1850	7	28	4.1	N/A	35848
RayTrace	2557	1	17	3.5	N/A	28930
GravityWars	2750	11	90	5.5	N/A	40194
Kermit	4400	43	51	8	N/A	42208
DataBase	25154	85	249	17.1	N/A	163710

REQUIREMENTS

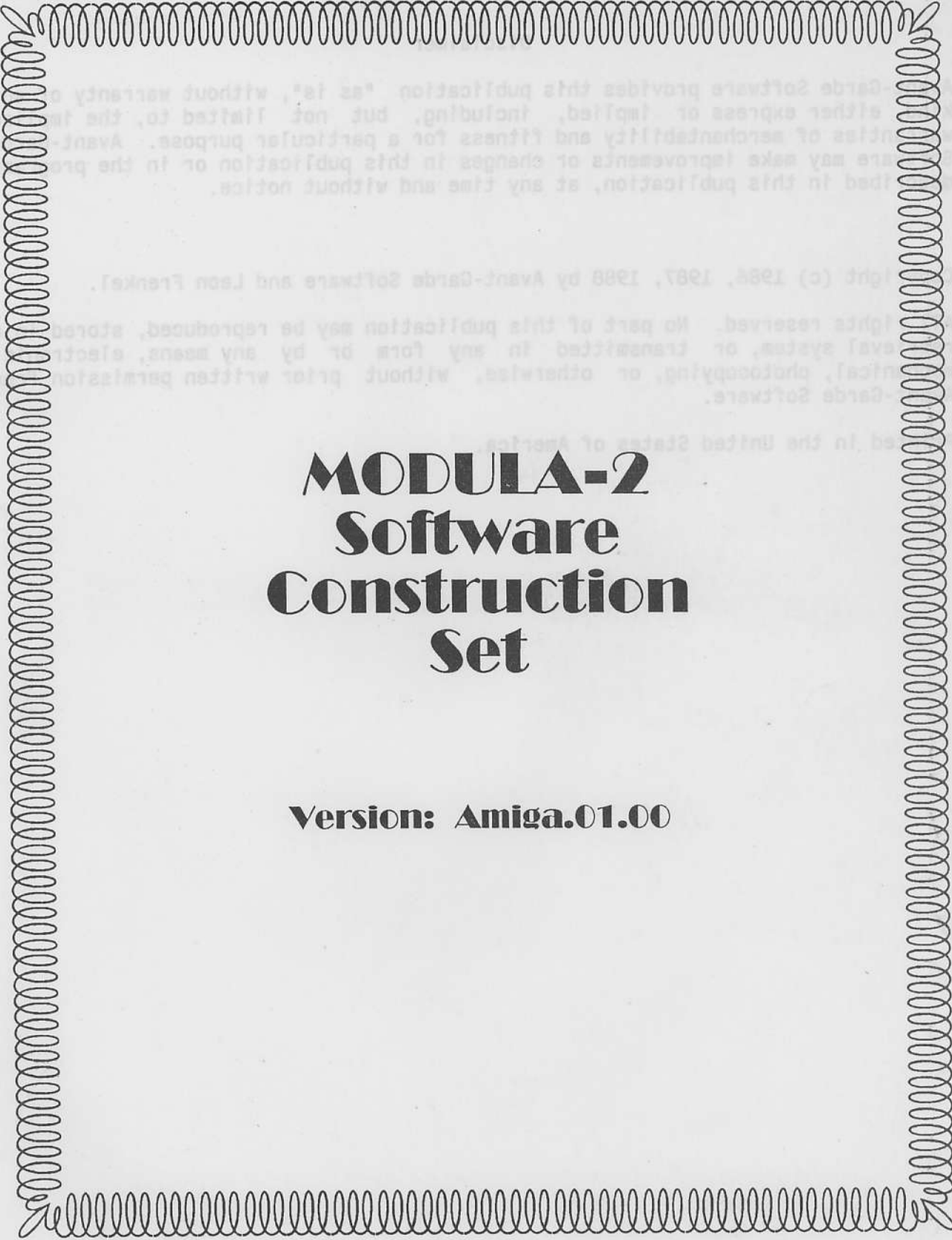
Amiga Computer A500/A1000/A2000,
One disk drive, 512K memory

A
V
A
N
T

G
A
R
D
E

S
O
F
T
W
A
R
E

Avant-Garde Software
2213 Woodburn
Plano, TX 75075
(214) 964-0260



MODULA-2 Software Construction Set

Version: Amiga.01.00

Disclaimer

Avant-Garde Software provides this publication "as is", without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Avant-Garde Software may make improvements or changes in this publication or in the programs described in this publication, at any time and without notice.

Copyright (c) 1986, 1987, 1988 by Avant-Garde Software and Leon Frenkel.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, or otherwise, without prior written permission from Avant-Garde Software.

Printed in the United States of America.

Preface

The **Modula-2 Software Construction Set** is a very efficient software development environment for the Commodore Amiga personal computer. The package can be used by both new and experienced Modula-2 programmers to create programs from the trivial to the very complex.

The package contains the necessary software tools and documentation to support the development of Modula-2 programs. The following are some of the items included in this package:

- * A very fast Modula-2 compiler which implements the entire Modula-2 language. The compiler provides a compilation speed of 10,000 lines per minute, with burst speeds of 30,000 lines per minute. The object code produced by the compiler is extremely fast and compact.
- * A lightning fast Modula-2 linker which binds together one or more separately compiled modules into an executable file. The executable file may be executed from the Command Line Interface (CLI), from WorkBench or from the EMACS editor.
- * An interactive full-screen editor based on EMACS. The editor supports editing of multiple files through one or more windows, with cutting and pasting between windows. There are approximately 150 commands available for performing many useful functions. The editor has been optimized for maximum speed, so most operations are performed in a fraction of a second. The editor has been seamlessly integrated with the compiler, linker and loader. This integration allows programs to be written, compiled, linked and run simply by pressing function keys in the editor.
- * As an alternative to the interactive environment each of the tools in the package may be used separately from the CLI. The compiler may be used from the CLI to batch compile a large number of files. The linker may be used to batch link a number of programs one after another. The EMACS editor is very useful for many different editing tasks, even light word processing is possible.
- * A very comprehensive collection of Modula-2 library modules is included. These libraries have been included so that the programmer can begin writing application programs immediately without having to reinvent the wheel. Each module has been written for maximum efficiency and has been rigorously tested for reliability.
- * A clean and efficient interface to all of the Amiga hardware and software is provided. Every function in the ROM Kernel, Intuition and AmigaDOS is available in Modula-2.
- * For compatibility with other Modula-2 implementations the following standard modules are included: InOut, RealInOut, LongInOut, FileSystem, Terminal, Storage, MathLib0, etc.
- * Many other miscellaneous utility library modules are provided to make programming in Modula-2 even easier.

- * A direct assembly language interface is available. A special utility is provided for converting from the standard Amiga object file format to the proprietary format used by the Modula-2 linker in this package.
- * A very comprehensive manual describing the tools included in the package and how to use them. The documentation for the libraries contains an example for every procedure defined. Full documentation for the functions in the ROM Kernel, Intuition and AmigaDOS is available in other technical documents for the Amiga.
- * An incredibly large collection of demonstration programs is included showing the use of almost every feature of the Amiga and Modula-2. The source code from any of the demonstration programs may be used in any user program without any additional cost. The example programs range from the very trivial to the very complex. The following is a very brief list of the programs included: **RayTrace** - a ray-tracing program which produces incredible 3-D Hold-and-Modify (HAM) pictures, **Draw** - a free hand drawing program, **DuM2** - a friendly alternative to the CLI, **GravityWars** - a one or two player strategy game, **Othello** - an interesting board game.
- * Comprehensive technical support is included with the price of this package. We will do our best to answer any questions you may have about using this package. Support is available via telephone or mail.
- * The product is constantly being improved and any new releases will be offered to registered users at a reasonable fee, depending on the nature of the upgrade. Bug fixes are not considered as upgrades and will be made available at cost of distribution or free if possible. Many enhancements are planned and will be implemented based on the responses we receive from users of the product.
- * Additional add-on products are available from us which further extend the usefulness of the package. The following additional products are currently available, with many other products in development:
 - Source Level Debugger
 - Simplified Amiga Libraries
 - "C" Language Libraries
 - IFF Libraries, Image Resource Utility
- * Third party support products will also be available in the near future to fill the needs of the wide variety of uses for this package.

Acknowledgments

Most of the software included in this package was written by Leon Frenkel. The text editor included in this package is based on Dave Conroy's MicroEmacs editor, it has been extensively enhanced by Mike Meyer, Mic Kaczmarczik, Leon Frenkel, Bob Larson and Dave Brower. The documentation for this package was written by Leon Frenkel. Some of the demonstration programs included in this package were written by Leon Frenkel, Mic Kaczmarczik, Scott Thompson, Ed Bartz, Richie Bielak, Steve Faiwischewski and many other Amiga hackers.

TABLE OF CONTENTS

Chapter 1	Introduction.....	3
Modula-2.....	3	
Modula-2 Software Construction Set.....	3	
Editor.....	3	
Compiler.....	3	
Linker.....	4	
Libraries.....	4	
Chapter 2	Installation.....	3
Introduction.....	3	
System Requirements.....	3	
Starting AmigaDOS.....	3	
Backing Up the Original Diskettes.....	3	
The Standard "Boot" Disk.....	3	
Installing M2SCS.....	4	
Compiler Configuration Utility.....	5	
Using M2SCS On a 512K Amiga.....	5	
Using M2SCS On a >512K Amiga.....	8	
The "Tiny" EMACS Editor.....	8	
Chapter 3	Tutorial.....	3
Editing.....	3	
Keyboard Command Abbreviations.....	4	
Cursor Commands.....	5	
Insertion/Deletion Commands.....	6	
File Commands.....	7	
Buffer Commands.....	8	
Window Commands.....	10	
Region Commands.....	11	
Compiling/Linking/Running.....	12	
Chapter 4	Modula-2 Compiler.....	3
Running the Compiler.....	3	
Compilation Errors.....	6	
Compiler Return Codes.....	7	
Separate Compilation and Version Control.....	8	
Compiler Switches.....	9	
Standard Procedures.....	11	
Pseudo Module SYSTEM.....	29	
Modula-2 Data Types.....	42	
Long Literal Constants.....	43	
Identifiers.....	44	
Size Limitations.....	44	
String Literals.....	45	
FORWARD Procedure.....	46	
CODE Procedures.....	46	
Run Time Error Checking.....	46	
Register Usage.....	47	
Short and Long Addressing.....	48	
Chapter 5	Modula-2 Linker.....	3
Running the Linker.....	3	
The Version Control System.....	4	
Symbolic Debugging.....	5	

Run-Time Support Library.....	5
Executable File Structure.....	5
Linker Limitations.....	6
Chapter 6 EMACS Editor.....	3
Running the Editor.....	3
The Editing Environment.....	3
EMACS Terminology.....	4
Keyboard Commands.....	6
Menu Commands.....	8
Mouse Commands.....	8
Overwrite Mode.....	10
Fill Mode.....	10
Delayed Prompts.....	10
Command Prefix Argument.....	11
Rebinding Keys.....	11
Expression Evaluation.....	12
Echo Line Prompt.....	13
Auto Completion of Names.....	13
Executing Commands by Name.....	14
Keyboard Macro.....	14
The EMACS Program Window.....	15
The File Requester.....	15
The Current Directory.....	17
Exact Filename Mode.....	17
The Modula-2 Compiler.....	18
The Modula-2 Linker.....	20
Running Modula-2 Programs.....	21
Command Descriptions by Topic.....	22
Cursor Commands.....	22
Window Commands.....	23
Buffer Commands.....	24
File Commands.....	25
Copy/Delete/Kill Commands.....	26
Search/Replace Commands.....	28
Case Commands.....	29
Key Binding Commands.....	30
Macro Commands.....	30
Fill Mode Commands.....	30
Expression Commands.....	31
Miscellaneous Commands.....	31
Mouse Commands.....	34
Display Setup Commands.....	35
Modula-2 Compiler Commands.....	36
Modula-2 Linker Commands.....	37
Modula-2 Run Commands.....	38
Chapter 7 Modula-2 Error Lister.....	3
Running the Error Lister.....	3
Chapter 8 Modula-2 Compiler Configuration Utility.....	3
Running the Configuration Utility.....	3
Chapter 9 Quick Load Utility.....	3
Creating a File List.....	3
Loading a Quick Load File.....	4
Making a Quick Load File.....	4

Chapter 10	Modula-2 Procedure Statistics Utility.....	3
	Running the Statistics Utility.....	3
Chapter 11	Symbol File Cross Reference Utility.....	3
	Running the Cross Reference Utility.....	3
Chapter 12	Object Module Generator Utility.....	3
	Running the Object Module Generator.....	3
Chapter 13	Assembly Language Interface.....	3
	Background Information.....	3
	Compiling the Definition Module.....	3
	Assembling the Implementation Module.....	3
	Linking the Object File.....	4
	Converting to an Object Module.....	4
	Example Conversion Batch File.....	5
	Example of an Assembly Language Module.....	5
Chapter 14	Module ASCII.....	3
	CharIsASCII.....	4
	CharIsControl.....	4
	CharIsPrintable.....	4
Chapter 15	Module Conversions.....	3
	ConvStringToNumber.....	4
	ConvNumberToString.....	5
Chapter 16	Module FileSystem.....	3
	Close.....	4
	Delete.....	5
	Lookup.....	6
	SetPos.....	7
	GetPos.....	8
	Length.....	9
	ReadWord.....	10
	WriteWord.....	11
	ReadChar.....	12
	WriteChar.....	13
Chapter 17	Module Heap.....	3
	ALLOCATE.....	4
	DEALLOCATE.....	5
	Available.....	6
	FreeHeap.....	7
Chapter 18	Module InitMathLib0.....	3
	OpenMathLib0.....	4
	CloseMathLib0.....	5
Chapter 19	Module InOut.....	3
	OpenInput.....	4
	OpenOutput.....	5
	OpenInputFile.....	6
	OpenOutputFile.....	7
	CloseInput.....	8
	CloseOutput.....	9
	Read.....	10
	ReadString.....	11

ReadInt.....	12
ReadCard.....	13
ReadWrd.....	14
Write.....	15
WriteLn.....	16
WriteString.....	17
WriteInt.....	18
WriteCard.....	18
WriteOct.....	18
WriteHex.....	18
WriteWrd.....	18
 Chapter 20 Module LargeSets.....	3
CreateSet.....	4
DeleteSet.....	5
SetInfo.....	6
SetRange.....	7
In.....	8
InclElement.....	9
ExclElement.....	10
InclRange.....	11
ExclRange.....	12
CopySet.....	13
Union.....	14
Diff.....	15
Intersection.....	16
SymmetricDiff.....	17
 Chapter 21 Module LongInOut.....	3
ReadLongInt.....	4
ReadLongCard.....	5
WriteLongInt.....	6
WriteLongCard.....	6
WriteLongOct.....	6
WriteLongHex.....	6
 Chapter 22 Module MathLib0.....	3
arccos.....	4
arcsin.....	4
arctan.....	4
cos.....	4
sin.....	4
tan.....	4
cosh.....	5
sinh.....	5
tanh.....	5
exp.....	6
ln.....	6
log.....	6
power.....	6
sqrt.....	6
real.....	7
entier.....	7
DegToRad.....	8
RadToDeg.....	8
 Chapter 23 Module RandomNumbers.....	3
Random.....	4

Chapter 24	Module RealConversions	3
	ConvStringToReal	4
	ConvRealToString	5
Chapter 25	Module RealInOut	3
	ReadReal	4
	WriteReal	5
Chapter 26	Module RunTimeErrors	3
	InstallErrorHandler	4
	RemoveErrorHandler	5
Chapter 27	Module Storage	3
	ALLOCATE	4
	DEALLOCATE	5
	Available	6
Chapter 28	Module Strings	3
	StringLength	4
	ConvStringToUpperCase	5
	CompareString	6
	CompareStringCAP	7
	CopyString	8
	ConcatString	9
	InsertSubString	10
	OverwriteWithSubString	11
	DeleteSubString	12
	ExtractSubString	13
	LocateSubString	14
	LocateChar	15
Chapter 29	Module System	3
Chapter 30	Module Terminal	3
	Read	4
	BusyRead	5
	ReadAgain	6
	Write	7
	WriteLn	8
	WriteString	9
Chapter 31	Module TermInOut	3
	Read	4
	ReadString	5
	ReadInt	6
	ReadCard	7
	Write	8
	WriteLn	9
	WriteString	10
	WriteInt	11
	WriteCard	11
	WriteOct	11
	WriteHex	11
Chapter 32	Definition Modules	3
	ADKHardware	4
	Alerts	5

AmigaDOS.....	8
AmigaDOSExt.....	14
AmigaDOSProcess.....	16
Areas.....	19
ASCII.....	21
AudioDevice.....	22
Blit.....	23
BootBlock.....	27
CIAHardware.....	28
CIAResource.....	30
ClipboardDevice.....	31
Clipping.....	32
CList.....	34
ConfigRegs.....	37
ConsoleDevice.....	40
ConsoleDevUnit.....	42
Conversions.....	43
Copper.....	44
CopperUtil.....	46
CustomHardware.....	47
DiskFont.....	49
DiskResource.....	51
Display.....	52
DMAHardware.....	53
Drawing.....	54
Exec.....	57
ExecBase.....	58
Expansion.....	60
FileHandler.....	63
FileSystem.....	65
GamePortDevice.....	67
Gels.....	68
Graphics.....	74
GraphicsBase.....	75
Heap.....	77
InitMathLib0.....	78
InOut.....	79
InputDevice.....	81
InputEvents.....	82
Interrupts.....	84
INTHardware.....	86
Intuition.....	87
IntuitionBase.....	111
IODevices.....	113
IODevicesUtil.....	116
KeyboardDevice.....	117
KeyMapResource.....	118
LargeSets.....	119
Layers.....	121
Libraries.....	124
Lists.....	126
LongInOut.....	128
MathLib0.....	129
Memory.....	131
MiscResource.....	134
NarratorDevice.....	135
Nodes.....	137
ParallelDevice.....	138

Ports.....	139
PortsUtil.....	141
PotgoResource.....	142
Preferences.....	143
PrinterBase.....	147
PrinterDevice.....	149
RandomNumbers.....	152
Rasters.....	153
RealConversions.....	155
RealInOut.....	156
Regions.....	157
Resident.....	159
Resources.....	160
RomLayers.....	161
RunTimeErrors.....	162
Semaphores.....	163
SerialDevice.....	165
Sprites.....	167
Storage.....	169
Strings.....	170
System.....	172
Tasks.....	174
TasksUtil.....	177
Terminal.....	178
TermInOut.....	179
Text.....	180
TimerDevice.....	182
TrackDiskDevice.....	183
Translator.....	185
Views.....	186
Workbench.....	189
Appendix A Error Messages.....	3
Modula-2 Compiler Error Messages.....	3
List of Compiler Errors.....	3
Description of Compiler Errors.....	7
Description of Non Compilation Errors.....	21
Description of Fatal Errors.....	22
Modula-2 Linker Error Messages.....	23
EMACS Editor Error Messages.....	25
General Errors.....	25
Modula-2 Compiler Errors.....	28
Modula-2 Linker Errors.....	30
Modula-2 Run Errors.....	32
Miscellaneous Errors.....	33
Modula-2 Error Lister Error Messages.....	34
Modula-2 Object Module Generator.....	35
Appendix B Command Summaries.....	3
Modula-2 Compiler.....	3
Modula-2 Linker.....	3
Modula-2 Error Lister.....	4
Modula-2 Compiler Configuration Utility.....	4
Modula-2 Symbol File Cross Reference Utility.....	4
Modula-2 Procedure Statistics Utility.....	4
Modula-2 Object Module Generator.....	4
Quick Loader Utility.....	5
EMACS Editor.....	5

Appendix C	Definition Modules Cross Reference.....	3
Appendix D	Conversion from "C" to Modula-2.....	3
	Comments.....	3
	Identifiers.....	3
	Integer Constants.....	3
	Character Constants.....	4
	Floating Point Constants.....	4
	String Constants.....	4
	lvalue.....	4
	Simple Data Types.....	5
	Pointer Type.....	6
	Array Type.....	6
	Structure Type.....	7
	Union Type.....	7
	Function Type.....	8
	Boolean Type.....	8
	Set Types.....	8
	Type Conversion.....	8
	Type Casting.....	9
	"if" Statement.....	9
	"while" Statement.....	9
	"do" Statement.....	10
	"for" Statement.....	10
	"switch" Statement.....	10
	"break" Statement.....	11
	"continue" Statement.....	11
	"return" Statement.....	11
	"goto" Statement.....	12
Appendix E	Conversion From TDI To M2SCS.....	3
	Compiler Differences.....	3
	Standard Modules.....	5
	Amiga Modules.....	6
Appendix F	Compatibility and Portability.....	3
	Modula-2 Compiler.....	3
	Standard Modules.....	3
	Non Standard Modules.....	4
Appendix G	Demonstration Programs.....	3
Appendix H	Glossary.....	3
Appendix I	Modula-2 Language Syntax.....	3
Appendix J	Bibliography.....	3

Chapter 1 Introduction.....	3
Modula-2.....	3
Modula-2 Software Construction Set.....	3
Editor.....	3
Compiler.....	3
Linker.....	4
Libraries.....	4

Chapter 1. Introduction.....	3
Module 2.....	3
Module 2 Software Construction Set.....	3
Editor.....	3
Compiler.....	3
Linker.....	4
Libraries.....	4

This chapter provides a brief introduction to the Modula-2 language and to the Modula-2 Software Construction Set (M2SCS).

Modula-2

Modula-2 is a general purpose high-level programming language suitable for solving a great variety of programming problems. Modula-2 is a descendant of PASCAL and was designed by Niklaus Wirth as a replacement for PASCAL.

The Modula-2 language is very easy to learn, especially for programmers familiar with PASCAL, "C" or another high-level language. The core language is small and very consistent. One thing that will be immediately noticeable is that a Modula-2 program listing can be very easy to read and understand even if the original author did not include any comments.

Modula-2 Software Construction Set

This package has been designed to provide a very efficient programming environment for using the Modula-2 language. It is well known that the programming environment being used greatly affects the productivity of the programmer. With this in mind, this package has been designed to provide a programmer with the most productive tools for programming available anywhere.

The major software tools included in this package are: an Editor, a Compiler, a Linker and Support Libraries. The following paragraphs present a brief explanation of each of these tools.

Editor

The editor allows text to be entered or modified in the computer. The editor is used to enter the source text of a Modula-2 program into the computer. The M2SCS editor is an EMACS style editor with a large set of commands for manipulating text. Any number of text files can be edited simultaneously, this allows fast cutting and pasting of text from one location to another. In addition, the editor serves as a convenient user interface to the compiler and linker.

Compiler

The compiler translates a Modula-2 source text into machine language, ready to be linked by the linker. The compiler is very efficient and will compile large programs very quickly. The compiler can be called from within the editor to provide instant feedback of any errors found in the Modula-2 source text.

Modula-2 Software Construction Set Reference Guide

Linker

The linker joins one or more separately compiled modules into an executable program. Once the linker has generated the executable file the program can be run to see if the results are correct. The linker can be called from within the editor.

Libraries

The libraries are pre-built and tested modules for performing many useful functions. Almost every Modula-2 program will utilize one or more of these libraries. This package includes an unusually large number of such libraries providing functions useful in almost any program and significantly reducing the amount of time required to write a program.

Editor

The editor allows text to be entered or modified in the computer. The editor is used to enter the source text of a Modula-2 program into the computer. The M2S2S editor is an EMACS style editor with a large set of commands for manipulating text. Any number of text files can be edited simultaneously. This allows fast cutting and pasting of text from one location to another. In addition, the editor serves as a convenient user interface to the compiler and linker.

Compiler

The compiler translates a Modula-2 source text into machine language ready to be linked by the linker. The compiler is very efficient and will compile large programs very quickly. The compiler can be called from within the editor to provide instant feedback of any errors found in the Modula-2 source text.

Chapter 2 Installation.....	3
Introduction.....	3
System Requirements.....	3
Starting AmigaDOS.....	3
Backing Up the Original Diskettes.....	3
The Standard "Boot" Disk.....	3
Installing M2SCS.....	4
Compiler Configuration Utility.....	5
Using M2SCS On a 512K Amiga.....	5
Using M2SCS On a >512K Amiga.....	8
The "Tiny" EMACS Editor.....	8

Chapter 2 Installation.....	2
Introduction.....	3
System Requirements.....	3
Starting AmigaDOS.....	3
Backing Up the Original Diskettes.....	3
The Standard "Boot" Disk.....	3
Installing MS2CS.....	4
Compiler Configuration Utility.....	5
Using MS2CS on a 512K Amiga.....	5
Using MS2CS on a 128K Amiga.....	6
The "Tiny" EMACS Editor.....	6

This chapter contains installation procedures which must be performed before beginning to use this package. This chapter assumes you are already familiar with the AmigaDOS Command Line Interface (CLI). If you are not familiar with the CLI refer to the AmigaDOS User's Manual or another reference book.

Introduction

The Modula-2 Software Construction Set has been designed to be easily installed on any Amiga configuration. Follow the instructions in each section of this chapter to properly install the software tools.

System Requirements

The following are the minimum requirements for running the Modula-2 Software Construction Set:

- * Amiga Computer (with 512K bytes of memory)
- * Kickstart/Workbench Version: 1.2 or higher

Significant performance benefits can be obtained by expanding the memory in the system and/or installing a hard-disk storage system.

Starting AmigaDOS

To use the M2SCS package you will need to use the CLI. Boot up the Amiga (refer to the documentation provided with the Amiga) and activate a CLI, when you get the "1>" prompt you are ready to proceed.

Backing Up the Original Diskettes

Before going any further make a backup of the original diskettes. This can be done using the "DiskCopy" command from the CLI. Make a backup copy of each diskette and put the original diskettes in a safe place.

The Standard "Boot" Disk

A pre-configured "Boot" disk is included to allow you to get a feel for the package without having to perform any installation or read the documentation. The "Boot" disk automatically loads the editor, compiler, linker and then loads in a small Modula-2 program "Hello.MOD". The sample program contains a comment which describes how to compile, link and run the program.

If you are using a system with more than 512K of memory spend a little time and setup the package to use the extra memory, the pay off in time saved compiling/linking is well worth it. The "Boot" disk was configured to allow a user to begin working immediately, but is not necessarily the most optimal

configuration possible. After you have satisfied your initial curiosity, read the documentation and setup the package for your specific hardware configuration.

Installing M2SCS

The installation procedure for the M2SCS package will vary depending on the hardware configuration being used. The information in this section is common to all configurations.

In order to work properly the tools in this package expect the following AmigaDOS device assignments:

- T: - Directory to be used for the compiler error log file
- M2L: - Directory which contains the Modula-2 libraries

The assignments should be performed in the "startup-sequence" file on the WorkBench diskette used in booting your system. The startup file may look like this:

```
ASSIGN T: RAM:T
ASSIGN M2L: DF1:M2L
```

The actual directories to which the "M2L:" and "T:" point may vary depending on the configuration of your system. If both assigns refer to a ram-based disk the software will operate significantly faster then if both are assigned to a floppy disk.

The tools included in this package should be placed in the "c:" directory of your disk. The directory containing the Modula-2 libraries assigned to "M2L:" must be on a disk always accessible to the compiler and linker. This directory contains a large number of files, two for each library module, one with the extension ".SBM" the other with the extension ".OBM". To save on disk space it may be useful to create a directory containing only the libraries which are actually used in your programs.

If memory permits, it is recommended that the libraries be copied to the ram disk when performing compile or link operations, this will improve the speed of those operations. The fastest way to copy the libraries to the ram disk is using the quick loader utility (QLoad). This program takes an archive file and copies its contents to a specified destination directory much quicker than if the same operation was done using the CLI "copy" command. Refer to the chapter on the QLoad utility for more information.

When you first boot AmigaDOS the default stack size is set to 4000 bytes. This stack size should be increased to 10000 bytes when running the compiler from the CLI. Demonstration and user created programs may also require additional stack space to function properly. To be safe it is recommended that you set the stack size to at least 10000 bytes in the startup-sequence file, this may be done with the following command:

```
STACK 10000
```

To summarize, the following commands should be added to your existing startup-sequence file in order for the tools in this package to function properly:

```
ASSIGN T: RAM:T
ASSIGN M2L: DF1:M2L
STACK 10000
```

Compiler Configuration Utility

Before you can begin using the compiler it may have to be configured according to your needs and the amount of memory available in your system. This configuration can be performed easily using the compiler configuration utility (M2Config). For further instructions refer to the chapter describing the M2Config utility.

The "Boot" disk contains a version of the compiler configured for a 512K Amiga. The "Tools" disk contains a version of the compiler configured for an Amiga with more than 512K of memory. Both compilers are identical with the exception that each has been configured with different system parameters. Either copy of the compiler may be used directly or reconfigured to fit your needs.

Using M2SCS On a 512K Amiga

The software in this package has been designed to work comfortably on an Amiga with 512K of memory. Most software development using this package will be performed while the EMACS editor and compiler are loaded into memory. In order to leave as much room as possible for development, the following suggestions are recommended.

The following is a short summary of the major points described in detail in the following text. If you have trouble understanding the long explanation, just do the following and you will achieve good results:

- * Do not use a ram disk or virtual disk.
- * Set the directory "T:" to floppy or hard disk and not to ram disk! This can be accomplished by adding the following to the startup-sequence: "ASSIGN T: df0:T". A directory called "T" should already exist on the designated disk prior to performing the ASSIGN, if the directory does not exist, create it using the "MakeDir df0:T" command. The disk on which the "T" directory is stored should not be write protected. The compiler writes out an error log file to the "T:" directory.
- * Close the original CLI window. By running EMACS using the "RunBack" command (supplied in this package) and then closing the CLI using "EndCLI". In the following example it is assumed the EMACS editor is in the "c:" directory.

```
RunBack c:M2Ed          ; run EMACS
EndCLI                  ; close CLI window
```

The "T:" directory should be directed to one of the floppy disks or the hard disk and a ram disk should not be used at all. Even when no files are present on the ram disk the system is using a significant amount of memory, so you should not activate the ram disk at all.

Another way to save memory is to use the "Tiny" version of the editor which has a few of the commands in the full-size editor removed to save space. The major item not found in the "Tiny" version is the file requester, instead files are selected at a prompt shown on the last line of the display. The decision to use the full-size or tiny version of EMACS is very subjective and depends on how well you can remember filenames on your disks.

The original CLI opened by AmigaDOS should be closed after EMACS is started, this can only be accomplished if EMACS is started using the "RunBack" command. If EMACS is started using the standard AmigaDOS "Run" command the original CLI will not close using the "EndCLI" command. The "RunBack" command performs a function similar to the "Run" command, however, it allows the original CLI to be closed successfully. If you need to execute a CLI command later you can open a new CLI from EMACS by typing Ctrl-2 or selecting the "New CLI" command from the menu. After you are done with the CLI, use the "EndCLI" command to close the CLI. You may continue to use EMACS while the new CLI is opened, you may also open additional CLI windows.

If you are going to be running the compiler only from within EMACS use a standard 4K stack when running EMACS, if you encounter any problems increase the stack size a bit. Note, EMACS allocates a stack for the compiler which is by default 10K.

By default EMACS uses a 20K stack when running user programs from the editor. For many programs this is too large and can be reduced to gain more space. It should be noted that the stack is allocated only when a user program is actually running and it does not effect the amount of available memory when no programs are running. The run stack size can be adjusted using the command "set-m2-run-stack-size" the command is available from the "Modula-2" menu.

Additional memory can also be saved by setting up the compiler to use a minimal amount of memory, this can be done using the M2Config utility and using the "set-m2-comp-opt" command from the "Modula-2" menu in EMACS. The M2Config utility makes a permanent change to the compiler configuration while the "set-m2-comp-opt" command only changes the configuration for the current session. A great deal of memory can be saved by reducing the compiler's heap size to a value less than 80K. However, this is only possible if the programs you wish to compile do not import the Intuition library, directly or indirectly. For example, a program which is file or text oriented may not need to use the Intuition library and could significantly reduce the amount of heap space required by the compiler.

If you follow the above suggestions, specifically, not using a ram disk and closing the CLI window, you can expect a very significant improvement in the amount of memory available to work in. The following table shows the approximate amount of free memory available after loading the editor/compiler/linker on a 512K system. The amount of free memory can be determined using the EMACS command "free-mem-avail" command from the menu.

- * Tiny Editor/Compiler/Linker/Run: 195000 bytes free
- * Full Editor/Compiler/Linker/Run: 181000 bytes free

When a compilation is started, the compiler allocates memory for its heap and internal buffers, so while the above specified memory is available during editing, part of the that memory is needed by the compiler while it is actually compiling. The amount of memory needed by the compiler depends on its configuration, you may examine and/or alter the amount of memory used by the compiler using the editor's "set-m2-comp-opt" command. If the compiler aborts due to a lack of memory try getting rid of text buffers which may be loaded but are not needed at the moment.

If disk space is important keep in mind that once the editor and the compiler are loaded it is no longer necessary to have the disk containing them in the drive. This fact may be especially important for a one disk system where you may wish to load the editor and compiler and then replace the Workbench disk with another disk containing the libraries and the source you wish to compile.

Using M2SCS On a >512K Amiga

On an Amiga with greater than 512K of memory significant improvements in performance can be obtained by using the additional memory in the following ways:

- * Copy the Modula-2 libraries onto the RAM disk. This will speedup both compiling and linking. The fastest way to copy the libraries to RAM is using the QLoad utility which can perform the entire operation in approximately 40 seconds, compared to using the AmigaDOS "Copy" command which could take 10 minutes. Further, by using a recoverable ram disk the load operation will only need to be performed once and if a system crash occurs the libraries will usually remain on the ram disk after rebooting the system.
- * Perform all compiles and links in RAM by either directing output to the RAM disk using the "-o" switch of the compiler and the "-o" switch of the linker, or simply keep the RAM disk as the default directory so that all output will default to the RAM disk. It is best to keep the original source files on floppy because even recoverable RAM disks are not 100% reliable and not much is gained by keeping the source files in RAM as opposed to floppies.

The "Tiny" EMACS Editor

Two versions of the EMACS editor are included on the "Tools" disk, "M2Ed" and "M2Ed.Tiny". The program "M2Ed" is the full-size editor as described in the documentation, the program "M2Ed.Tiny" is a smaller version of the editor which saves about 15K bytes and may be useful to users with 512K of memory by providing more work space. The tiny editor is almost identical to the full-size editor, however, in the title bar the word "Tiny" appears. The following items are NOT available in the "Tiny" version of the editor:

- * File Requester Window is not available, as is the associated command:
 - file-req-mode
- * Startup File ".emacs" is not supported, the following commands associated with evaluating are not available:
 - eval-current-buffer
 - eval-expression
 - load

- * Commands to change the color and rendition of text are not available:
 - set-mode-background
 - set-mode-foreground
 - set-mode-rendition
 - set-text-background
 - set-text-foreground
 - set-text-rendition
- * File backup feature controlled by the command are not available:
 - make-backup-files
- * Prefix region commands are not available:
 - set-prefix-string
 - prefix-region
- * Parenthesis matching commands are not available:
 - blink-matching-paren
 - blink-matching-paren-hack

Chapter 3 Tutorial.....	3
Editing.....	3
Keyboard Command Abbreviations.....	4
Cursor Commands.....	5
Insertion/Deletion Commands.....	6
File Commands.....	7
Buffer Commands.....	8
Window Commands.....	10
Region Commands.....	11
Compiling/Linking/Running.....	12

Chapter 3 Tutorial.....	3
Editing.....	3
Keyboard Command Abbreviations.....	4
Cursor Commands.....	5
Insertion/Deletion Commands.....	6
File Commands.....	7
Buffer Commands.....	8
Window Commands.....	10
Region Commands.....	11
Compiling/Linking/Running.....	12

The purpose of this tutorial chapter is to familiarize you with the basic operations of editing, compiling, linking and running Modula-2 programs. It is assumed at this point that the installation of the software has been completed successfully. If the installation has not been completed return to the chapter on installation before continuing with the tutorial. It is beyond the scope of this tutorial to teach you how to program in Modula-2, for that you will need to read a book which describes the Modula-2 language specifically, a short list of Modula-2 books is presented in the appendix of this manual.

To achieve the maximum benefit from the tutorial you should be sitting in front of your computer actually trying the commands being discussed in the tutorial.

For extra convenience a function key strip has been included which describes the default functions key assignments in EMACS. The function key strip should be placed at the top of the keyboard as you will need to refer to it during this tutorial.

After completing the tutorial you will be able to edit, compile, link and run a Modula-2 program. If after finishing the tutorial you are not completely comfortable with the operation of the system, repeat the tutorial again. After you have mastered the basics of using the tools, read the rest of the documentation to find out about additional features available in the package.

Editing

Before starting the tutorial on the editor it is suggested the first part of the chapter on the editor be read. The first part of the chapter describes the basic concepts of text editing, as well as, the special terminology specific to the EMACS editor.

For this part of the tutorial you will need to run EMACS and read in the text file "EditTutorial.doc" which contains some arbitrary text on which you may try the new editing commands you will learn. To begin using the editor, run the editor program "M2Ed" with the text file to be used for the tutorial as a command line argument: "M2Ed EditTutorial.doc". The text file "EditTutorial.doc" is located on the "Boot" disk. If necessary, copy the "EditTutorial.doc" file to your own work disk before running the editor. If the text file is not in the current directory, then change the current directory using the CLI's "CD" command to the directory which contains the file.

After some disk activity the editor will open a window displaying as its contents the specified text file. If the editor opens a window but the window is empty, this indicates that the editor could not find the specified file in the specified directory. Exit the editor by clicking on the "close-box" gadget at the upper left hand corner of the window. Check to make sure the text file is accessible and that you have typed the name in correctly, when you are sure everything is setup correctly try the command again.

Keyboard Command Abbreviations

In this tutorial and elsewhere in this manual EMACS keyboard commands will be shown using the following abbreviations:

- * Ctrl-<Char> - To produce the specified command press and hold the key labeled <CTRL> while typing the specified character.
- * Esc-<Char> - To produce the specified command type the key labeled <ESC>, release the key, then type the specified character.

Examples:

Ctrl-n = Press and hold the control key and type "n".

Esc-v = Type the escape key then type "v".

EMACS can be forced to abort any partially entered command or prompt by entering Ctrl-g. For example, if you have begun a command which begins with <Esc> and then decided to abort the command after typing the <Esc> key, pressing Ctrl-g will abort the command.

While working with EMACS if you find that you have forgotten the meaning of a specific key or combination of keys, pressing the <HELP> key followed by the key combination will display the name of the function associated with the key combination, on the echo line.

Example:

Pressing <HELP> will display the message "Describe key briefly:", now typing the right arrow key will display the message "Right runs the command forward-char", on the echo line.

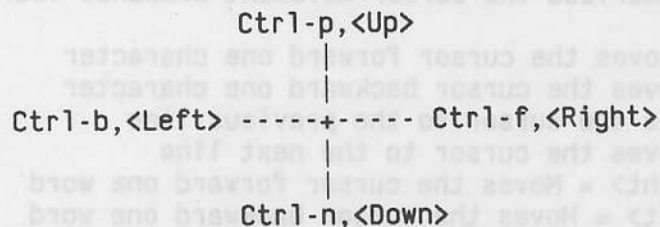
Throughout this tutorial commands will usually be referred to by the key combination necessary to execute the command. An alternative way of executing many of the commands is to use the pull-down menus. The menus are accessible by pressing down and holding the right mouse button and then moving the mouse to position the "pointer" on the specified menu title which will cause a menu of commands to appear below the menu title. To select a specific command move the "pointer" to the corresponding menu item and release the right mouse button. To help you master the keyboard commands faster, most of the menu commands show the default keyboard command which can be used to perform the same function. The keyboard command displayed in the menus use the following abbreviations:

C- = Control, same as Ctrl-

M- = Esc, same as Esc-, ("M" stands for META)

Cursor Commands

The basic cursor control commands allow you to move the cursor forward, backward, up or down. Two sets of keys are available for performing the basic cursor movement operations, the arrow keys and control keys. The following diagram illustrates the basic cursor movement commands:



Moving forward from the last character of the line will move the cursor to the first character of the next line. Moving backward from the first character of the line will place the cursor at the last character of the previous line.

When the cursor is moved above the top line or below the bottom line of the window, the contents of the window will scroll down or up respectively. After the window scrolls up or down, the line to which the cursor was being moved will be positioned in the center of the display.

If a line is longer than the width of the display, the right edge of the display will contain the character "\$" to indicate there are additional characters which are not visible. The cursor may be positioned on the invisible characters, but the solid cursor block will not move beyond the right edge of the display.

Since moving a character at a time can be slow, EMACS has commands for moving a word at a time. EMACS considers a word to be a sequence of alpha-numeric characters terminating on a white space, the end of line or a punctuation character. To move the cursor forward a word at a time use Esc-f or <Shift>-<Right>. To move the cursor backward a word at a time use Esc-b or <Shift>-<Left>.

Additional commands are available for moving the cursor to either end of a line in one quick step. To move to the beginning of the line use Ctrl-a. To move to the end of the line use Ctrl-e.

At any given moment only a small portion of a buffer is visible in the window. To view the previous page of the buffer use Esc-v. To view the next page of the buffer use Ctrl-v.

Finally, there are commands for moving to either end of the buffer instantaneously. To move to the beginning of the buffer use Esc-<. To move to the end of the buffer use Esc->.

An alternative to moving the cursor with keyboard commands is to use the mouse. Move the mouse to the desired position and click the left button once, the cursor will move to the new position. This only works for text currently visible in the window.

The following table summarizes the cursor movement commands learned so far:

- * Ctrl-f or <Right> = Moves the cursor forward one character
- * Ctrl-b or <Left> = Moves the cursor backward one character
- * Ctrl-p or <Up> = Moves the cursor to the previous line
- * Ctrl-n or <Down> = Moves the cursor to the next line
- * Esc-f or <Shift>-<Right> = Moves the cursor forward one word
- * Esc-b or <Shift>-<Left> = Moves the cursor backward one word
- * Ctrl-a = Moves the cursor to the first character of the line
- * Ctrl-e = Moves the cursor to the last character of the line
- * Ctrl-v = Moves the cursor to the next page of the buffer
- * Esc-v = Moves the cursor to the previous page of the buffer
- * Esc-< = Moves the cursor to the first character of the buffer
- * Esc-> = Moves the cursor to the last character of the buffer

Before continuing with the tutorial, spend a few minutes experimenting with the cursor control commands until you are comfortable with their operation.

Insertion/Deletion Commands

To insert text into the buffer simply begin typing characters and they will be inserted at the cursor position. When you wish to begin a new line of text press the <RETURN> key and a new line will be created with the cursor positioned on the first character of the line.

While typing, if you notice that the previous character typed is incorrect, pressing the <BackSpace> key or its equivalent Ctrl-h, will delete the previous character and move the cursor backwards by one character. To delete the character that is under the cursor, press the key or its equivalent Ctrl-d, any characters remaining on the line will move backwards by one character to fill the empty space.

A "newline" character is stored at the end of each line of text to indicate the termination of the line. Deleting the "newline" character will cause the next line to merge and become part of the current line. If a line becomes too long to be displayed in the window a "\$" character will appear in the rightmost column of the display indicating that the line contains additional characters which are not visible on the display.

When EMACS is directed to perform a delete on something larger than a character it stores the deleted text in a "kill buffer". The text can be restored by "yanking" the text from the "kill buffer".

For instance, to delete a word, position the cursor to the right of the word you wish to delete then type Esc-d or Esc-. The specified word will be deleted from the text buffer. To restore the word at another position in the text, move the cursor to the new position and press Ctrl-y to "yank" the word from the "kill buffer" into the text buffer at the cursor position. A word to the left of the cursor may be deleted using the command Esc-<BackSpace>.

Another very useful command is "kill-line" invoked by Ctrl-k, this command causes the characters from the cursor position to the end of the line to be deleted. The "newline" character at the end of the line is not deleted.

If the cursor is not moved between multiple kill operations, the characters from each kill will be appended to the kill buffer. For example, an entire paragraph may be killed one line at a time and then the cursor may be moved to a new position where the paragraph may be inserted with one "yank" operation.

The following table summarizes the delete commands learned so far:

- * Ctrl-d or = delete the character under the cursor
- * Ctrl-h or <BackSpace> = delete the character to the left of the cursor
- * Esc-d or Esc- = delete a word to the right of the cursor
- * Esc Ctrl-h or Esc-<BackSpace> = delete a word to the left of the cursor
- * Ctrl-k = delete the characters from the cursor position to the end of the line
- * Ctrl-y = yank the characters previously stored in the "kill buffer" into the text buffer at the current cursor position

Before continuing with the tutorial, spend a few minutes experimenting with the insertion and deletion commands until you are comfortable with their operation.

File Commands

You will now be introduced to the EMACS commands which relate to file operations. EMACS allows files to be read into buffers for editing or viewing purposes. After a buffer has been edited, the contents of the buffer may be saved into the same file or a different file.

The "find-file" command creates a new buffer and reads the contents of the specified file into the buffer. The "find-file" command may be invoked by pressing the Ctrl-x Ctrl-f key combination, <F9> on the function key strip or selecting the command from the pull-down menu. If the specified file has already been loaded into EMACS, the buffer containing the file will be displayed rather than loading the file in from disk again.

Whenever EMACS requires the name of a file to be specified, a file requester window will pop-up on the display. The title of the file requester window describes the function to be performed with the selected filename. The "File Name" gadget will be active, the name of a file may be typed followed by <RETURN> or clicking the [ACCEPT] gadget. To examine the files in the

directory, click the mouse button in the file display window. The scroll-bar may be used to examine the contents of the entire directory, if it is larger than the display window. To select a file in the window, double-click on the name of the file or click on it once and then click on the [ACCEPT] gadget. To abort the file operation, click on the close-box or click on the [CANCEL] gadget.

An alternative way of specifying the name of a file is to answer a prompt shown on the last line of the window. The prompt is not as user friendly as the file requester, but may be faster for some experienced users. The input method for filenames may be toggled, with the command "file-req-mode" on the "File" menu.

Read in a text file from disk using the "find-file" command. If the specified file is not found on disk EMACS will display the message "(New file)" indicating that a new file is being edited. If the file is found then the new buffer containing the text file will be displayed. The contents of the buffer may now be edited using any of the EMACS commands.

After the buffer has been edited, the contents of the buffer may be saved to disk using the "save-buffer" command, Ctrl-x Ctrl-s or <F10>. To save the buffer under a different name than was used to read the file use the command "write-file", Ctrl-x Ctrl-w or Shift-<F10>. A new file name may now be specified, the contents of the buffer will be saved to the specified file. If the specified file already exists on disk, the old file will be deleted and the new file will be stored in its place.

The following table summarizes the file commands learned so far:

- * Ctrl-x Ctrl-f or <F9> = Read file into buffer or edit a new file
- * Ctrl-x Ctrl-s or <F10> = Save contents of current buffer to disk
- * Ctrl-x Ctrl-w or Shift-<F10> = Save contents of current buffer to a specified file

Before continuing with the tutorial, spend a few minutes experimenting with the file management commands introduced in this section. When you are comfortable with the operation of these commands continue with the tutorial.

Buffer Commands

An extremely useful feature of EMACS is that more than one file may be edited simultaneously without requiring the previous file to be saved and a new one loaded. This has many advantages including cutting and pasting between files, this will be explored later in this tutorial.

The number of files which may be edited simultaneously is only limited by the amount of free memory available in the system, since each file has to be in memory.

While a file is loaded in EMACS it is referred to as a "buffer". A buffer has an associated buffer name which is usually the same as the name of the file, but does not have to be the same. A buffer usually has an associated filename with a full directory path to that file. A buffer may have no filename associated with it, this may be useful for some temporary editing which you do not wish to save to disk.

At any given moment there is one buffer called the "current" buffer, this is the buffer in which the cursor or "point" is located. Switching from one buffer to another is very easy using the command "switch-to-buffer", Ctrl-x b. A prompt will appear asking for the name of a buffer to switch to, a default buffer to switch to will be displayed. To switch to the default buffer type <RETURN>, otherwise type the name of the buffer and then type <RETURN>. Whenever EMACS issues a prompt for a buffer name an auto-completion feature is available. To select a given buffer it is only necessary to type the first few characters which make it obvious which name is being requested. Typing <Space> will cause the current word to be completed if possible, typing <TAB> will try to complete the entire name and typing <RETURN> will try to complete the entire name and then perform the function. If EMACS can't auto-complete it will display the message "[No match]" if no name exists consisting of the characters typed, or "[Ambiguous]" to indicate there are several possibilities and more characters have to be typed to determine which name is being requested. The "switch-to-buffer" command may also be used to create new buffers by responding to the prompt with a name of a buffer not currently defined in EMACS. In either case, after this command is successfully completed the specified buffer will be displayed in the current window.

Since EMACS can have many buffers loaded at one time a command is available to look at a list of all the buffers loaded into EMACS. The buffer list is generated by the command "list-buffers", Ctrl-x Ctrl-b. The buffer list command causes the current window to split into two windows and the buffer list appears in the second window. The buffer list is generated in a special buffer which is always named "*Buffer List*", the buffer list contains a line of data for each buffer in the system. The line shows the size of the buffer, the filename path for the buffer and if the buffer has been modified.

Buffers may be deleted when no longer necessary using the command "kill-buffer", Ctrl-x k. This command prompts for the name of a buffer to be killed (deleted) from EMACS, the default will always be the name of the current buffer. To kill the current buffer type <RETURN>. If the buffer has been changed but not saved to a file, then a prompt will appear asking you to confirm the kill operation on the selected buffer.

The following table summarizes the buffer commands learned so far:

- * Ctrl-x b = switch to a specified buffer or create a new buffer, the specified buffer becomes the "current" buffer
- * Ctrl-x Ctrl-b = display a list of buffers currently available in EMACS
- * Ctrl-x k = delete the specified buffer, the contents of the buffer are discarded

Before continuing with the tutorial, spend a few minutes experimenting with the buffer management commands introduced in this section. When you are comfortable with the operation of these commands continue with the tutorial.

Window Commands

A very powerful feature of EMACS is the ability to display many different buffers on the same display simultaneously. Even different parts of a given buffer can be displayed simultaneously. To save memory and to provide more convenience, the buffers are displayed in mini-windows which are part of the main EMACS window. This approach is much more efficient than having many separate Amiga windows cluttering up the display. From this point on these EMACS mini-windows will be referred to simply as windows, but they should not be confused with the overlapping Amiga windows like the one being used for the entire EMACS display.

New windows are opened by splitting the current window, using the command "split-window-vertically", Ctrl-x 2. Both windows will now be displaying the same buffer. To change the view in the current window, move the cursor to a different part of the buffer, using the cursor movement commands. To display a completely different buffer in the current window the command "switch-to-buffer", Ctrl-x b may be used as described earlier.

The cursor may be moved from window to window using the commands "next-window", Ctrl-x n, or "previous-window", Ctrl-x p. The cursor position is remembered for each window and upon returning to a window the cursor will be at its previous position. To quickly move the cursor to any window on the display, position the mouse at the desired location and click the left mouse button.

Additional windows may be created by splitting a window one or more times. Any window at least 3 lines in height may be split. To delete the current window, use the command "delete-window", Ctrl-x Ø. To restore the display to a one window display, use the command "delete-other-windows", Ctrl-x 1.

Each window has three gadgets on its mode line for manipulation of the window, the functions performed by these gadgets may also be performed from the keyboard or via menu selections. The following gadgets are shown on the mode line of each window:

```
[*] = delete window
[v] = enlarge window
[^] = shrink window
```

To perform one of the above operations, position the mouse pointer on the gadget and click the left mouse button.

The following table summarizes the window commands learned so far:

```
* Ctrl-x 0 = delete the current window
* Ctrl-x 1 = return editor display to one window
* Ctrl-x 2 = split the current window into two windows
* Ctrl-x n = move cursor to next window (down) on the display
* Ctrl-x p = move cursor to previous window (up) on the display
* Gadget [*] = delete window
* Gadget [v] = enlarge window
* Gadget [^] = shrink window
```

Before continuing with the tutorial, spend a few minutes experimenting with the window management commands introduced in this section. When you are comfortable with the operation of these commands continue with the tutorial.

Region Commands

As you may remember from the "terminology" section of the editor chapter, there are two cursors in EMACS, a visible cursor called the "point" and an invisible cursor called the "mark". The text between the "point" and the "mark" is called the "region".

A number of useful operations may be performed on the text in the "region". The basic operations are: delete, copy, convert to upper case or convert to lower case. Before any of the operations may be performed, the "region" must be specified. Move the "point" to the position which is to serve as one end of the "region", set the "mark" at this position. The "mark" is set using Ctrl-@ (it is not necessary to use shift). The "mark" may also be set by double clicking on a character. After the "mark" has been set, move the cursor to the opposite end of the text you wish to designate as a "region".

Now that the region has been selected, one of the region operations may be performed. To delete the region use Ctrl-w, the region will be deleted from the buffer and placed into the kill buffer. As you may recall the contents of the kill buffer may be inserted at the current cursor position using the "yank" command, Ctrl-y. Using these two commands allows very convenient cutting and pasting text of between one or more buffers. To place a copy of the selected region into the kill buffer, without deleting the region, use the command Esc-w. To convert all of the lower case letters in a region to upper case use the command Ctrl-x Ctrl-u. To convert all of the upper case letters in a region to lower case use the command Ctrl-x Ctrl-l.

The "mark" may also be used for another purpose which is not directly associated with the "region". The "mark" may be used as a way of remembering the current cursor position so it can be returned to later. The command Ctrl-x Ctrl-x swaps the "mark" with the "point" thereby moving the cursor to a previously selected position. For example, if the "mark" is set in the middle of the buffer and the "point" is at the beginning of the buffer, the cursor will be moved to the middle of the buffer. Typing Ctrl-x Ctrl-x again will move the "point" to the beginning of the buffer and restore the "mark" to the center of the buffer.

Before continuing with the tutorial, spend a few minutes experimenting with the region commands introduced in this section. When you are comfortable with the operation of these commands continue with the tutorial.

Compiling/Linking/Running

This part of the tutorial will teach you how to interactively compile, link and run Modula-2 programs from within the EMACS editor. For this part of the tutorial you will need to read in the text file "CompTutorial.mod", this file is located on the "Boot" disk. It may be necessary to copy this file to your own work disk before continuing with the tutorial. Read the file into an EMACS buffer using the find-file command.

The contents of the text file "CompTutorial.mod" is a very simple Modula-2 program which has some intentionally placed errors. These errors will prevent this program from being successfully compiled until the errors are fixed.

To compile the program "CompTutorial.mod" move the cursor into the window which is associated with the buffer that contains the text file and press the <F2> key. Since this is the first time we are using the compiler since loading EMACS, the message "Loading Modula-2 Compiler..." will be displayed, and the compiler will be read in from disk. Hereafter the compiler will not be reloaded and will be available for all future compile operations. If an error message is displayed indicating that the compiler was not loaded successfully, then this means that the installation may not have been performed correctly or there is insufficient memory to load the compiler. Either of these problems will have to be resolved before continuing with the tutorial. The appendix of this manual contains all the possible error messages along with explanations which may help solve the problem. Also, check the installation procedure to make sure everything was done properly.

As the compilation is performed line numbers will quickly flash on the screen indicating the current line being compiled. Within a second or so the compilation will be completed and the message "Source Compiled, 4 Errors Found." will be displayed on the echo line. The message indicates that the compiler has detected 4 errors in the source code which was compiled.

The next step is to locate the errors, determine the reason for the errors and fix the errors. EMACS has a built in error tracking facility which can position the cursor right on or near the location of the error. To move the cursor to the first error, press <F1>, the cursor will be positioned on the line:

```
MyStr := "Hi!";
```

The title bar of the EMACS window will display the compiler error message:

Error 50: identifier not declared or not visible

This message will also appear on the echo line of the window. However, as soon as any EMACS command is performed the error message will disappear from the echo line, the message in the title bar will remain.

The error message indicates that the identifier "MyStr" is not defined. The identifier which should have been specified is "MyString" defined a few lines earlier in the program. So, to fix the error, change "MyStr" to "MyString" in this line. The new line should be:

```
MyString := "Hi!";
```

Now that this error has been fixed, we have two choices: we can try to compile the file again or we can continue to fix any other errors remaining in the file. Assuming we decide to continue fixing errors, pressing <F1> will move the cursor to the location of the next error in the file. The cursor will be positioned on the line:

```
WriteString(MyStr);
```

The title bar of the EMACS window will display the compiler error message:

Error 50: identifier not declared or not visible

The error message indicates that the identifier "MyStr" is not defined. The identifier which should have been specified is "MyString" defined a few lines earlier in the program. Again, to fix the error change "MyStr" to "MyString" in this line. The new line should be:

```
WriteString(MyString);
```


In this particular program both errors were related, but this is just a coincidence and in many other cases errors will not be related.

Pressing <F1> again to position the cursor on the next error causes the echo line to display the message "End of Error List! (Returning to First Error)". This message indicates that we have stepped through all the known errors and the error tracker has returned to the first error in the file. It will not always be possible to step through the errors again, because error positions are sometimes lost when the text containing the error is edited. This is the reason why there were 4 errors found during compilation, but only two were encountered while stepping through the errors.

Now that all the known errors have been fixed, attempt to compile the file again by pressing <F2>. This time after compilation the following message will be displayed on the echo line:

```
Object Module Generated. Code:xxxx UData:xxxx IData:xxxx Total:xxxx
```

This indicates that the file was compiled successfully and an object module has been generated. The positions where "xxxx" is shown will contain actual values, giving the size of these program components. If an error message is displayed, then a mistake was probably made when fixing the two errors, examine the lines in question to make sure they have been changed exactly as specified above.

Congratulations! You have now successfully fixed the errors in this program and compiled the program.

The next step to be performed before you can see the results of your effort is to link the program, even though the user program consists of only one module, the link step is still required to link in the necessary system modules.

To make the development cycle more automated specify the name of the "main" module, the root of the Modula-2 program. Press <F6>, the prompt "Main Module:" will appear on the echo line, in response type "CompTutorial". Specifying the name of the "main" module this way, eliminated the need to retype it every time you wish to link or run the program, EMACS will remember this name until the name is changed or you exit EMACS.

Now that EMACS knows the name of the "main" module, press <F3> to link the program. Assuming the package has been installed correctly, within a few seconds the link operation will be done. While linking, the names of the modules being linked will be displayed very briefly at the bottom of the display.

Assuming the link operation was successful the size of the program will be displayed.


```
Code:xxxx UData:xxxx IData:xxxx Init:xxxx Total:xxxx
```

Now that you have successfully compiled and linked the test program, it is probably a good idea to save the modified source back to disk. This way if the program you are developing causes the system to crash, you will not loose any of your work. It is also a good idea to save your work often while you are writing programs to avoid loosing your work in case of a power failure or some other uncontrollable occurrence.

Finally, all that is left is to run the program to see if it performs the intended function correctly. To run the test program, press <F4>. A window with the title "CompTutorial" will open and the test program will print the message:

```
You Have Successfully Completed The Tutorial!  
Hi!
```

A bit further down in the window the following message will be displayed:

```
== Press <RETURN> To Exit ==
```

This message indicates that the test program has exited and now the system is waiting for you to type <RETURN> so that it may close the window. This is done, so that you have a chance to see any output produced by the program, before the window is closed. You may now press <RETURN> to close the window. It should be noted that the window was automatically opened by EMACS and is not the responsibility of the user program. If the same program is run from the CLI this window will not be opened.

You have now successfully completed the tutorial. You should have a very good feel for how to do all of the steps in the development cycle. Many other commands and options are available in EMACS, but your current knowledge will allow you to get started.

```
Code:xxxx IDate:xxxx IDate:xxxx IDate:xxxx IDate:xxxx
```

Now that you have successfully compiled and linked the test program, it is probably a good idea to save the modified source back to disk. This way if the program you are developing causes the system to crash, you will not lose any of your work. It is also a good idea to save your work often while you are writing programs to avoid losing your work in case of a power failure or some other uncontrollable occurrence.

Finally, all that is left is to run the program to see if it performs the intended function correctly. To run the test program, press <F4>. A window with the title "Computorial" will open and the test program will print the message:

```
You Have Successfully Completed The Tutorial!  
Hi!
```

A bit further down in the window the following message will be displayed:

```
-- Press <RETURN> To Exit --
```

This message indicates that the test program has exited and now the system is waiting for you to type <RETURN> so that it may close the window. This is done so that you have a chance to see any output produced by the program before the window is closed. You may now press <RETURN> to close the window. It should be noted that the window was automatically opened by EMACS and is not the responsibility of the user program. If the user program is run from the CLI this window will not be opened.

You have now successfully completed the tutorial. You should have a very good feel for how to do all of the steps in the development cycle. Many other commands and options are available in EMACS, but your current knowledge will allow you to get started.

Chapter 4	Modula-2 Compiler	3
Running the Compiler		3
Compilation Errors		6
Compiler Return Codes		7
Separate Compilation and Version Control		8
Compiler Switches		9
Standard Procedures		11
Pseudo Module SYSTEM		29
Modula-2 Data Types		42
Long Literal Constants		43
Identifiers		44
Size Limitations		44
String Literals		45
FORWARD Procedure		46
CODE Procedures		46
Run Time Error Checking		46
Register Usage		47
Short and Long Addressing		48

Chapter 4: Modula-2 Compiler.....	5
Running the Compiler.....	5
Compilation Errors.....	6
Compiler Return Codes.....	7
Separate Compilation and Version Control.....	8
Compiler Switches.....	9
Standard Procedures.....	11
Pseudo Module SYSTEM.....	19
Modula-2 Data Types.....	22
Long Literal Constants.....	23
Identifiers.....	24
Size Limitations.....	24
String Literals.....	25
FORWARD Procedures.....	26
CODE Procedures.....	26
Run Time Error Checking.....	26
Register Usage.....	27
Short and Long Addressing.....	28

This section describes the Modula-2 compiler. The compiler is a complete implementation of Modula-2 as described in the book "Programming in MODULA-2.", 3rd Edition by Niklaus Wirth, creator of the language.

The compiler accepts as its input a source file containing one of the following types of modules:

- * Definition Module - The global specification part of a library module. The filename has the extension ".DEF".
- * Implementation Module - The implementation part of a library module. The filename has the extension ".MOD".
- * Program Module - The module which serves as the root of a Modula-2 program. The filename has the extension ".MOD".

After a successful compilation of a source file, the compiler will produce one of the following types of modules:

- * Symbol Module - The global specification of a library module in compact binary form. This type of module is produced as a result of compiling a definition module. The filename has the extension ".SBM".
- * Object Module - The native machine code produced as a result of compiling an implementation or program module. The filename has the extension ".OBM".
- * Reference Module - An optionally generated module containing debugging information, for use by a source level debugger. This module may be generated as a result of compiling an implementation or program module. The filename has the extension ".RFM".

Running the Compiler

WARNING: Before running the compiler make sure that at least 10000 bytes of stack space is allocated. This can be done from the CLI by using the command "STACK 10000". Failure to do this may cause the compiler to crash while compiling. It should also be noted that if the compiler crashes with a stack size set to 10000 bytes, the stack size should be increased. Increasing the stack size will almost always solve the problem of the compiler mysteriously crashing. This note does not directly apply when using the compiler from within the EMACS editor, because EMACS will automatically allocate a stack for the compiler.

Modula-2 Software Construction Set Reference Guide

The name of the compiler program is "M2". The syntax for running the compiler is as follows:

M2 [Options] [SourceFile] [SourceFile] ...

If no source files are found on the command line, the compiler will prompt the user to enter the name of a source file. The prompt "Source File [DEF|MOD]:" will be displayed. At this point the name of a source file may be entered or the <RETURN> key may be pressed to exit the compiler. If a source filename is entered, the compiler will attempt to compile the file. After the file has been compiled the prompt will reappear again.

If the compiler is invoked with one or more source filenames specified on the command line, the compiler will proceed to compile each source file in the order listed. After compiling all the source files the compiler will return to the CLI. To abort compilation before the compiler finishes compiling the source files, press CTRL-C. This will cause the compiler to return to the CLI after it finishes compiling the current source file.

The compiler has many parameters which may be configured permanently using the M2Config utility. For more information refer to the chapter on the M2Config utility.

In addition to source filenames the compiler accepts numerous command line options. A command line option begins with the character "-" followed by a letter specifying the option. An option may be a toggle switch or it may be followed by an additional argument. Options may be listed in any order and may be listed before, after or between source filenames. Options specified on the command line override the default configuration set by the M2Config utility. For toggle switches, the option will be toggled to the opposite state from the default configuration. The following is a list of the available compiler options and what they do:

* **-s <DirectoryPath>**

Specifies the name of a directory to be searched for symbol files. The option may be used more than once to specify additional directories to be searched. By default the "M2L:" directory is searched automatically, the directory should contain the system library modules which all programs will need. The path name must be terminated by the character "/" or ":".

Example: M2 -s RAM: -s DF0:MyLibs/

* **-o <DirectoryPath>**

Specifies the name of the directory to which files generated by the compiler should be written. The path name must be terminated by the character "/" or ":".

Example: M2 -o RAM:

- * **-r**
Toggle the generation of run-time range checking code. Range checking applies to assignments of sub-range variables and to array subscripts.
- * **-v**
Toggle the generation of run-time overflow checking code. Overflow checking applies to INTEGER and LONGINT arithmetic.
- * **-g**
Toggle the generation of a reference module as a result of compiling an implementation or program module. The reference module contains debugging information for use by a source level debugger. The reference file may also be read by the linker to add symbols to an executable file. These symbols can be used by a symbolic debugger. For instructions on how to include the symbols in an executable file refer to the symbolic debugging section of the linker documentation.
- * **-l**
Toggle the generation of 32-bit or 16-bit addressing for global variables. Both the DEFINITION and IMPLEMENTATION part of a library module should be compiled with this option in the same state.
- * **-k**
Toggle the generation of a zero version control key rather than the randomly generated version control key. When compiling a definition module, instead of assigning the module a randomly generated version control key, the key will be set to zero. This option should only be used when compiling a new System module. It should never be used to compile any other module.
- * **-h <Size>** **<Size> = 10000..MaxRam**
Specifies the number of bytes to be allocated for the heap. If a heap overflow error occurs, increasing the size of the heap will resolve the problem. If not enough memory exists to increase the heap, split the module into two or more modules and compile separately.
Example: M2 -h 150000
- * **-i <Size>** **<Size> = 1000..32766**
Specifies the number of bytes to be allocated for the identifier buffer. If an identifier buffer overflow error occurs, increasing the size of the identifier buffer will eliminate the problem.
Example: M2 -i 24000
- * **-c <Size>** **<Size> = 1000..32766**
Specifies the number of bytes to be allocated for the code buffer. If a code buffer overflow error occurs, increasing the size of the code buffer will eliminate the problem.
Example: M2 -c 16000

* **-a <Size>** **<Size> = 1000..32766**

Specifies the number of bytes to be allocated for the address relocation buffer. If an address relocation buffer overflow error occurs, increasing the size of the buffer will eliminate the problem.

Example: M2 -a 20000

* **-b <Size>** **<Size> = 1000..32766**

Specifies the number of bytes to be allocated for the constant buffer. If a constant buffer overflow error occurs, increasing the size of the buffer will eliminate the problem.

Example: M2 -b 8000

* **-d <Size>** **<Size> = 512..MaxRam**

Specifies the number of bytes to be allocated for the disk buffer. This option can be used to change the size of the buffer the compiler uses for reading and writing files. Specifying a larger disk buffer size may, in some cases, increase the speed of compilation. For each file opened by the compiler a disk buffer of the specified size will be allocated.

Example: m2 -d 4096

In the process of compiling a source file, the compiler will display the filenames of the symbol files that it is importing. The files that are being imported are preceded by "<-" indicating input. The files that are being generated by the compiler are preceded by "->" to indicate that they are being output.

If the compiler cannot find a symbol file that is needed to compile the current source file, it will print ">>Error Opening or Reading Symbol File!" after the name of the symbol file. This error usually occurs when the modules of a program are not compiled in the correct order. To fix the problem recompile the modules again in the proper order based on the module dependencies.

When the compiler successfully compiles an implementation or program module, it will display the program size in bytes:

Code:xxxxx UData:xxxxx IData:xxxxx Total:xxxxx

Code - bytes of machine code, statements

UData - bytes of uninitialized data, global variables

IData - bytes of initialized data, strings

Total - total module size, sum of the above three numbers

Compilation Errors

Before the compiler begins compilation it creates an error log file called "M2.err". The error log file is created in the "T:" directory. When an error is encountered during compilation, the error and its position is recorded in the error log file. The compiler will display "Errors in Source File!" if it finds any errors in the source file. After returning to the CLI, you can run the

error lister program "M2Err" to see exactly where each error occurred and a description of the error. For more information refer to the error list chapter of this manual.

If the compiler is used to compile more than one source file during one invocation of the compiler, the errors for all of the source files compiled will be placed in the error log file. The error lister will display the errors for each source file in the order in which they were encountered.

Each time the compiler is invoked the old error log file is deleted. Therefore, after several invocations of the compiler, only the errors from the last invocation will be displayed by the error lister.

Compiler Return Codes

When the compiler is invoked from inside a CLI batch file, it generates a return code indicating if any problems were encountered. The following is a list of possible return codes:

- 0 - No problems were encountered
- 5 - Source file(s) had compilation error(s)
- 20 - Invalid command line arguments
- 20 - Insufficient memory
- 20 - Error log file not opened
- 20 - Source file(s) not found or failed to open
- 20 - Output file(s) failed to open

The following is an example batch file which uses the return code from the compiler to perform an appropriate action. For more information on batch files refer to the AmigaDOS user's guide.

MakeModule:

```
.Key module/a
M2 <module>.mod
IF WARN
M2Err
ELSE
M2Lk <module>
ENDIF
```

Example of usage: Execute MakeModule Draw

This batch file attempts to compile the specified module. If the compiler returns an error code ≥ 5 (WARN) then the error lister program is executed displaying the compilation error messages. If the compiler returns an error code of < 5 then the compilation has been successful so the linker is invoked to produce an executable file.

Separate Compilation and Version Control

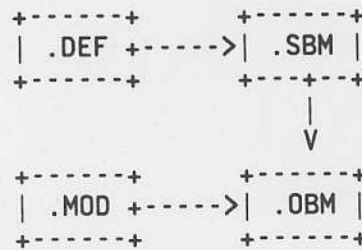
In many older languages, it is not uncommon to accidentally link together old and new versions of object files. This may pass successfully through a linker, but the resulting executable file will not work. A lot of time may be wasted determining the reason for the strange behavior of the program.

The problem described above can never occur in Modula-2, because Modula-2 has a built in version control system. This system will prevent incompatible modules from being linked together. The version control system is the responsibility of the compiler and the linker. In this section the version control system and how it relates to the compiler will be discussed. For additional information refer to the linker section of this manual.

The compiler generates a symbol module file as a result of compiling a definition module. In the symbol file is a randomly generated key (identification number). Later, when the implementation module is compiled, the corresponding symbol module file is read and the same key assigned to the symbol module is used for the object module. If the implementation module is changed later, the same key will again be used. The same key is used because changing the implementation part of a module will not directly effect any other module. However, when the definition module is recompiled, the resulting symbol file will be assigned a new key. At this point the symbol file will be incompatible with the old object module, because they have different keys. The next time another module imports your new symbol module, it will associate that module with the key found in this module. If the compiler imports another symbol module which had imported the previous version of your symbol module, the compiler will report a key mismatch error. This prevents a disastrous situation from occurring. Therefore, always recompile the implementation module after the definition module is compiled, as well as, recompiling the modules which depend on that definition module.

A version conflict may not be detected by the compiler because it does not directly import any other symbol modules which have imported the older version of the symbol module. In such cases, the version conflict will be detected by the linker. The above stated solution of recompiling the dependent modules will resolve the problem. When all the version control keys are correct the linker will generate an executable file.

The following diagram illustrates the relationship of the source files and the compiler generated files:



Compiler Switches

A compiler switch is a special compiler command imbedded in comments in the source file. The compiler switch has the following syntax:

```

Switch = <Letter><"+"|"-">
Usage = "(*$" [Switch]{"," Switch} "*)"

```

To indicate that the comment contains one or more compiler switches a "\$" character must follow the "(" without any spaces in between. The <Letter> specifies the compiler switch to be affected. The <Letter> is followed by a "+" to turn on the compiler switch or a "-" to turn off the compiler switch. To specify additional compiler switches on the same line follow the switch by a "," character followed by the next switch. No spaces are allowed between any of the characters. The following compiler switches are supported:

\$R Enable or disable the generation of range checking code.
The following is an example of run-time range checking errors:

```

(*$R+*)
VAR
  x: [0..9];
  y: ARRAY [10..20] OF CHAR;
  z: CARDINAL;
BEGIN
  ...
  z := 100;
  x := z; (* error: value outside of sub-range bounds *)
  y[z] := "A"; (* error: index outside of array bounds *)
  ...

```

Modula-2 Software Construction Set Reference Guide

\$V Enable or disable the generation of overflow checking code.
The following is an example of a run-time overflow checking error:

```
(* $V+ *)
VAR
  x: INTEGER;
  y: INTEGER;
BEGIN
  ...
  x := 30000;
  y := 30000;
  x := x + y; (* error: overflow *)
  ...
```

\$D Enable or disable the generation of code to make local copies of dynamic value parameters. By disabling this option, the compiler will not generate the code necessary to make a local copy of "ARRAY OF" parameters that are supposed to be passed by value, it will not have any effect on VAR parameters. This option should be used very carefully as it may cause valid Modula-2 programs to operate improperly. This option may be used when speed of execution and more compact code is very important. In the following example this option is used to produce a more efficient procedure for determining the length of a string. This use of the option is safe because the original string is only read, never written to.

```
(* $D- *)
PROCEDURE StrLength(s: ARRAY OF CHAR): CARDINAL;
VAR length: CARDINAL;
BEGIN
  length := 0;
  WHILE s[length] # 0C DO
    INC(length);
  END;
  RETURN(length);
END StrLength;
(* $D+ *)
```

\$L Enable or disable the use of long absolute addressing to address global variables. By default the 16-bit address register relative mode is used to address global variables. If this switch is used for a procedure which is defined in the definition module then the \$L switch should be specified in both the DEFINITION and IMPLEMENTATION module. This switch should only be used by experienced programmers.

Examples:

```
(*R-,V-  disable range and overflow checking *)  
(*R+,V+  enable range and overflow checking *)
```

Standard Procedures

The following is a list of the standard procedures available in this implementation of Modula-2. A brief description of each procedure is given and an example of its use is presented. For additional information on standard procedures refer to a reference text on Modula-2. In this section there are several references to a scalar type, the following types are scalar: Enumeration, Subrange, BOOLEAN, CHAR, CARDINAL, INTEGER, LONGINT, LONGCARD.

Modula-2 Software Construction Set Reference Guide

ABS(x: INTEGER/LONGINT/REAL): INTEGER/LONGINT/REAL;

Return the absolute value of the input parameter. The input parameter must be of type INTEGER, LONGINT or REAL. The result is of the same type as the input parameter.

VAR

s, t: REAL;

i, j: INTEGER;

BEGIN

...

s := ABS(s + t);

i := ABS(j);

...

CAP(ch: CHAR): CHAR;

If the input parameter is a lower case letter then convert it to upper case, otherwise, return the same character.

VAR

 c: CHAR;

BEGIN

 ...
 IF CAP(c) = "Y" THEN

 ...
 ELSIF CAP(c) = "N" THEN

 ...
 END;

 ...

Modula-2 Software Construction Set Reference Guide

CHR(x: Type): CHAR;

Return the character with the ordinal number x. The input parameter must be a scalar type.

```
VAR
  i: CARDINAL;
  ch: CHAR;
BEGIN
  ...
  ch := CHR(10);
  ch := CHR(i + 30);
  ...
```

DEC(VAR x: Type);

This procedure is equivalent to $x := x - 1$, x must be a scalar type. Using this procedure produces more efficient code than the above stated equivalent form. Using this procedure may make the source code easier to read and understand.

```
VAR
  i: CARDINAL;
  j: CHAR;
BEGIN
  ...
  DEC(i);
  DEC(j);
  ...
```

DEC(VAR x: Type; n: INTEGER/CARDINAL/LONGINT/LONGCARD);

This procedure is equivalent to $x := x - n$, x must be a scalar type. Using this procedure produces more efficient code than the above stated equivalent form. Using this procedure may make the source code easier to read and understand.

```
VAR
  i: CARDINAL;
  j: LONGCARD;
BEGIN
  ...
  DEC(i, 10);
  DEC(j, i * 10);
  ...
```

Modula-2 Software Construction Set Reference Guide

EXCL(VAR x: Type; n: Type);

This procedure is equivalent to $x := x - \{n\}$, x must be a set type and n must an element of the set. Using this procedure produces more efficient code than the above stated equivalent form. Using this procedure may make the source code easier to read and understand.

```
VAR
  colors: SET OF [0..7];
  num: CARDINAL;
BEGIN
  ...
  EXCL(colors, 4);
  EXCL(colors, num + 2);
  ...
```

FLOAT(x: INTEGER/CARDINAL/LONGINT): REAL;

This procedure converts a value of type INTEGER, CARDINAL or LONGINT to floating point. In some Modula-2 implementations the type CARDINAL may be the only type of parameter accepted.

VAR

x: REAL;

a: CARDINAL;

b: INTEGER;

c: LONGINT;

BEGIN

...

x := FLOAT(a + 10);

x := FLOAT(b);

x := FLOAT(c * 100);

...

Modula-2 Software Construction Set Reference Guide

HALT;

This procedure will cause the program to terminate execution immediately. This procedure may be used to exit the program when an event has occurred which makes it impossible to continue. Another use of this procedure is to exit the program from a deeply nested procedure.

BEGIN

...

IF (ErrorFlag) **THEN**

 WriteString("Fatal Error Encountered!");

HALT; (* terminate program *)

END;

...

HIGH(x: Type): Type;

Return the highest index of a specified array, the result type is the same as the index type of the array. The input parameter to this procedure must be a static or dynamic array variable.

```
PROCEDURE DoString(str: ARRAY OF CHAR);
```

```
VAR
```

```
  buf: ARRAY [1..30] OF CHAR;
```

```
  h: CARDINAL;
```

```
BEGIN
```

```
  h := HIGH(buf); (* returns 30 *)
```

```
  h := HIGH(str); (* returns (depends on input array length) *)
```

```
END DoString;
```

Modula-2 Software Construction Set Reference Guide

INC(VAR x: Type);

This procedure is equivalent to $x := x + 1$, x must be a scalar type. Using this procedure produces more efficient code than the above stated equivalent form. Using this procedure may make the source code easier to read and understand.

```
VAR
  i: CARDINAL;
  j: CHAR;
BEGIN
  ...
  INC(i);
  INC(j);
  ...
```

INC(VAR x: Type; n: INTEGER/CARDINAL/LONGINT/LONGCARD);

This procedure is equivalent to $x := x + n$, x must be a scalar type. Using this procedure produces more efficient code than the above stated equivalent form. Using this procedure may make the source code easier to read and understand.

```
VAR
  i: CARDINAL;
  j: LONGCARD;
BEGIN
  ...
  INC(i, 10);
  INC(j, i * 10);
  ...
```

INCL(VAR x: Type; n: Type);

This procedure is equivalent to $x := x + \{n\}$, x must be a set type and n must be an element of the set. Using this procedure produces more efficient code than the above stated equivalent form. Using this procedure may make the source code easier to read and understand.

```
VAR
  colors: SET OF [0..7];
  num: CARDINAL;
BEGIN
  ...
  INCL(colors, 4);
  INCL(colors, num + 2);
  ...
```

Modula-2 Software Construction Set Reference Guide

MAX(x: Type): Type;

Return the maximum value of a specified type. The result is of the same type as the input parameter. The input parameter may be any scalar or floating point type.

TYPE

```
colors = (red, green, blue);  
range = [10..20];
```

VAR

```
a: INTEGER;  
b: CARDINAL;  
c: LONGCARD;  
d: REAL;  
e: colors;  
f: range;
```

BEGIN

```
a := MAX(INTEGER);      (* 32767 *)  
b := MAX(CARDINAL);     (* 65535 *)  
c := MAX(LONGCARD);     (* 4294967295 *)  
d := MAX-REAL);         (* 9.22337177E+18 *)  
e := MAX(colors);       (* blue *)  
f := MAX(range);        (* 20 *)
```

MIN(x: Type): Type;

Return the minimum value of a specified type. The result is of the same type as the input parameter. The input parameter may be any scalar or floating point type.

TYPE

 colors = (red, green, blue);
 range = [10..20];

VAR

 a: INTEGER;
 b: CARDINAL;
 c: LONGCARD;
 d: REAL;
 e: colors;
 f: range;

BEGIN

a := MIN(INTEGER);	(* -32768 *)
b := MIN(CARDINAL);	(* 0 *)
c := MIN(LONGCARD);	(* 0 *)
d := MIN-REAL);	(* -9.22337177E+18 *)
e := MIN(colors);	(* red *)
f := MIN(range);	(* 10 *)

Modula-2 Software Construction Set Reference Guide

ODD(x: INTEGER/CARDINAL/LONGINT/LONGCARD): BOOLEAN;

Return TRUE if the input parameter is an odd value or return FALSE if the input parameter is an even value. This procedure is equivalent to the expression $x \text{ MOD } 2 \neq 0$.

VAR

a: INTEGER;

c: LONGCARD;

f: BOOLEAN;

BEGIN

...

f := ODD(a * 13);

f := ODD(c);

...

ORD(x: Type): CARDINAL;

Return the CARDINAL with the ordinal number of the input parameter, x must be a scalar type.

```
VAR
  c, d: CHAR;
  e: (red, green, blue);
  a: CARDINAL;
BEGIN
  ...
  a := ORD(c);
  a := ORD(e);
  a := ORD(c) + ORD(d);
  ...
```

Modula-2 Software Construction Set Reference Guide

SIZE(x: Type): CARDINAL;

If the input parameter is a type then the result is the number of bytes required for a variable of the given type. If the input parameter is a variable then the result is the number of bytes of storage used by the variable.

TYPE

Node = RECORD x, y, z: REAL; END;

VAR

s: CARDINAL;

n: Node;

a: ARRAY [0..9] OF Node;

BEGIN

...

s := SIZE(Node);

s := SIZE(n);

s := SIZE(a);

s := SIZE(n[3]);

...

TRUNC(x: REAL): CARDINAL;

This procedure converts a floating point value to CARDINAL. The conversion is performed by truncating the fractional part of the floating point value.

VAR

x: REAL;

c: CARDINAL;

BEGIN

...

c := TRUNC(x);

c := TRUNC(x * 40.4);

...

VAL(t: Type; x: Type): Type;

This procedure is equivalent to a type transfer function, the following are equivalent VAL(CARDINAL, "A") and CARDINAL("A"). The first parameter specifies the resulting type and the second parameter is the expression to be coerced to the new type. This procedure may be used to make instances of type transfer easier to identify when reading the source code.

```
VAR
  a: CARDINAL;
  b: INTEGER;
  c, d: CHAR;
BEGIN
  ...
  a := VAL(CARDINAL, c) - VAL(CARDINAL, d);
  c := VAL(CHAR, b);
  ...
```

Pseudo Module SYSTEM

Processor-dependent features of Modula-2 are imported from the module SYSTEM. The module SYSTEM is unlike other modules in that it only exists within the compiler. SYSTEM should not be confused with **System** which is a normal module containing the run-time support for a Modula-2 program.

Most of the procedures defined in the module SYSTEM are implemented as inline code for purposes of efficiency. Since these procedures do not require a subroutine call, they are much more efficient than if the same function was performed by a user defined procedure.

The module SYSTEM in this implementation of Modula-2 can be considered to have the following definition module:

DEFINITION MODULE SYSTEM;

TYPE

 BYTE;
 WORD;
 LONGWORD;
 ADDRESS = POINTER TO BYTE;

PROCEDURE ADR(VAR x: Type): ADDRESS;
PROCEDURE TSIZE(x: Type): CARDINAL;
PROCEDURE SHORT(x: LONGINT): INTEGER;
PROCEDURE LONG(x: Type): LONGINT; or LONG(high, low: INTEGER): LONGINT;
PROCEDURE SHIFT(x: Type; n: INTEGER): Type;
PROCEDURE REG(reg: CARDINAL): LONGINT;
PROCEDURE SETREG(reg: CARDINAL; value: Type);
PROCEDURE INLINE(x: CARDINAL/INTEGER,...);

END SYSTEM.

A detailed description of each type and procedure defined in the pseudo module SYSTEM follows.

BYTE

The type BYTE represents the smallest unit of addressable storage, on a 68000 computer. A BYTE is defined as 8-bits and may represent values between 0 and 255. No operations may be directly performed on this type, other than assignment. A variable or procedure parameter of type BYTE may be assigned an expression of any type which is exactly 8-bits long. This allows a CHAR, ENUMERATION or a user defined set to be assigned to a BYTE directly. If a procedure parameter is declared as an ARRAY OF BYTE then the actual parameter may be of any type. An array of BYTE will be mapped onto the actual parameter. For example accessing element zero of an array will return the first byte of the actual parameter. If an operation needs to be performed on a variable of type BYTE, it must first be coerced into another type such as CARDINAL. It should also be noted that a BYTE may reside at any address odd or even. The following examples illustrate how BYTE can be used.

```

VAR
  flag: BYTE;
  ch  : CHAR;
  bits: SET OF [0..7];
  val : CARDINAL;
BEGIN
  ...
  flag := ch;
  flag := bits;
  flag := BYTE(CARDINAL(flag) + val);
  val  := TSIZE(BYTE); (* = 1 *)
  ...

```

WORD

The type WORD is defined as two bytes, containing 16-bits, therefore, it can only be used to represent values between 0 and 65535. No operations may be directly performed on this type, other than assignment. A variable or procedure parameter of type WORD may be assigned an expression of any type which is exactly 16-bits long. This allows an INTEGER, CARDINAL, BITSET, subrange or a user defined set to be assigned to a WORD directly. If a procedure parameter is declared as an ARRAY OF WORD the actual parameter may be any type at least 16-bits in length. An array of WORD will be mapped on top of the parameter regardless of the type of parameter being passed. If an operation needs to be performed on a variable of type WORD, it must first be coerced into another type such as CARDINAL. Finally, a WORD must be aligned on a word boundary (an even address). This is a requirement of the 68000 processor, if a read or write operation is performed on a WORD at an odd address, an address error exception will be generated by the processor. The following examples illustrate how WORD can be used.

```

VAR
  count: WORD;
  size : CARDINAL;
  bits : SET OF [0..12];
  mask : BITSET;
  year : [1980..2001];
  breg : POINTER TO WORD;
BEGIN
  ...
  breg^ := count;
  count := size;
  count := bits;
  count := mask;
  count := year;
  count := CARDINAL(count) * size;
  size := TSIZE(WORD); (* = 2 *)
  ...

```

LONGWORD

The type LONGWORD is defined as four consecutive bytes, containing 32-bits, therefore, it can be used to represent values between 0 and 4,294,967,295. No operations may be directly performed on this type, other than assignment. A variable or procedure parameter of type LONGWORD may be assigned an expression of any type which is exactly 32-bits long. This allows a LONGINT, LONGCARD, pointer, procedure variable or a user defined set to be assigned to a LONGWORD directly. A parameter declared ARRAY OF LONGWORD maps an array of LONGWORD elements over the actual parameter, regardless of the structure of the actual parameter. If an operation needs to be performed on a variable of type LONGWORD, it must first be coerced into another type such as LONGCARD. Finally, a LONGWORD must be aligned on a word boundary, an even address. This is a requirement of the 68000 processor, if a read or write operation is performed on a LONGWORD at an odd address, an address error exception will be generated by the processor. The following examples illustrate how LONGWORD can be used.

```

VAR
  lword: LONGWORD;
  size : LONGINT;
  bits : SET OF [0..31];
  breg : POINTER TO WORD;
BEGIN
  ...
  lword := size;
  lword := bits;
  lword := breg;
  size := TSIZE(LONGWORD); (* = 4 *)
  ...

```

ADDRESS

The type ADDRESS is defined as a pointer to a BYTE. This type is useful for performing address calculations and manipulating pointers to objects of different types. The type ADDRESS has the unique property of being compatible with any user defined pointer type, in addition for purposes of address calculation it is also compatible with LONGCARD. One common use of ADDRESS is as a parameter to a block of memory which contain different types of data. The following example demonstrates how to use the ADDRESS type.

```

VAR
  heapPtr: ADDRESS;
  winPtr: POINTER TO Window;
  cPtr   : ADDRESS;
  ch     : CHAR;

PROCEDURE Alloc(size: LONGCARD): ADDRESS;
BEGIN
  INC(heapPtr, size);
  RETURN heapPtr;
END Alloc;

BEGIN
  ...
  winPtr := Alloc(LONGCARD(TSIZE(Window)));
  cPtr^ := ch; (* assign CHAR to BYTE *)
  ...

```

ADR(VAR x: Type): ADDRESS;

The ADR procedure returns the absolute address of a specified object. The object may be a variable, procedure, or a string constant. If the object is a variable it may be of any type and may be defined as either a global or local variable. A string constant is assigned a memory location by the compiler and is therefore associated with an address. The address associated with a procedure is that of the first instruction of a procedure. The following examples illustrate the usage of the procedure ADR.

```
VAR
  x : CARDINAL;
  a : ADDRESS;
  p : POINTER TO ARRAY [0..255] OF CHAR;
BEGIN
  ...
  a := ADR(x);
  a := ADR(p[x]);
  a := ADR("Hello World!");
  ...
```

TSIZE(x: Type): CARDINAL;

The TSIZE procedure returns the size in bytes of a specified type identifier. This will be the same as the size of a variable of the same type. The following examples illustrate the usage of the procedure TSIZE.

```
TYPE
  str = ARRAY [0..255] OF CHAR;
  rec = RECORD x,y,z: CARDINAL; END;
```

```
VAR
  sz : CARDINAL;
BEGIN
  ...
  sz := TSIZE(str);
  sz := TSIZE(rec);
  ...
```


Modula-2 Software Construction Set Reference Guide

SHORT(x: LONGINT): INTEGER;

The procedure SHORT converts a LONGINT or LONGCARD to an INTEGER result. This operation is performed by truncating the high-order bits. Range checking is ignored for this operation.

```
VAR
  si : INTEGER;
  li : LONGINT;
BEGIN
  ...
  si := SHORT(li);
  ...
```

LONG(x: Type): LONGINT; or LONG(high, low: INTEGER): LONGINT;

The procedure LONG has two forms. The first form is selected by specifying only one argument, this argument may be of any scalar type. The result is a LONGINT representation of the argument. The second form is selected by specifying two arguments, both arguments must be INTEGER or CARDINAL. The result is a LONGINT with the high-order bits containing the first argument and the low-order bits containing the second argument. The following example demonstrates the usage of both forms of this procedure.

```
VAR
  x,y: CARDINAL;
  val: LONGINT;
  ch : CHAR;
BEGIN
  ...
  val := LONG(ch);
  val := LONG(x);
  val := LONG(x,y);
  ...
```

SHIFT(x: Type; n: INTEGER): Type;

The SHIFT procedure is used to shift left or right the bits of the argument passed to it. The first argument to SHIFT is a scalar type such as CARDINAL, LONGCARD, etc. The second argument is an INTEGER specifying the number of bits to shift and in which direction. If the second argument is positive then a left shift is performed. If the second argument is negative then a right shift is performed. The result of the procedure is the shifted value, the type of the result is the same as the type of the first argument. It should be noted that if the type being shifted is a signed type (i.e. INTEGER, LONGINT), the shift will be arithmetic and if the type is unsigned (i.e. CARDINAL, LONGCARD), a logical shift will be performed. The difference between the two types of shifts is in the handling of the sign bit. The following example demonstrates use of the SHIFT procedure.

```
VAR
  b,c : BITSET;
  x,y : INTEGER;
BEGIN
  ...
  b := SHIFT(c,4); (* shift left by 4 bits *)
  b := SHIFT(c,-1); (* shift right by 1 bit *)
  x := SHIFT(x, y); (* shift x by y bits left or right *)
  ...
```

```
REG(reg: CARDINAL): LONGINT;
```

The procedure REG causes the contents of a specified 68000 processor register to be returned as a LONGINT value. The argument to this procedure is a constant number between 0 and 15. If the number is between 0 and 7 it specifies a data register D0-D7 and if the number is between 8 and 15 it specifies an address register A0-A7. The following example demonstrates the use of the REG procedure.

```
VAR
  status : LONGINT;
  stackPtr: ADDRESS;
BEGIN
  ...
  status := REG(7); (* get D7 *)
  stackPtr := ADDRESS(REG(15)); (* get SP (reg A7) *)
  ...
```

SETREG(reg: CARDINAL; value: Type);

The procedure SETREG performs the opposite operation from the procedure REG, it allows the user to load a value into a specified 68000 processor register. The first argument to the procedure is a constant INTEGER specifying the register to be loaded, 0 to 7 or 8 to 15. The register number corresponds to D0..D7 and A0..A7 respectively. The second argument is the value to be loaded into the register. The value may be of any type which is exactly 32-bits long. The specified example uses the procedure SETREG.

```
CONST D4 = 4; A7 = 15;
```

```
VAR
```

```
  oldStackPtr: ADDRESS;
```

```
BEGIN
```

```
...
```

```
  SETREG(A7, oldStackPtr); (* restore the old stack ptr *)
```

```
  SETREG(D4, 1000000D);
```

```
...
```

INLINE(x: CARDINAL/INTEGER,...);

The **INLINE** procedure is useful for inserting machine language instructions into a Modula-2 procedure. The arguments to **INLINE** are **CARDINAL** or **INTEGER** values to be stored into the instruction stream, beginning at the current location. There is no limit to the number of arguments that can be specified, separated by commas. The following example demonstrates how **INLINE** may be used.

```
CONST
  BreakPt = 4AFCH; (* ILLEGAL instruction *)
BEGIN
  ...
  a := b;
  INLINE(BreakPt); (* Force debugger to stop here! *)
  c := d;
  ...
```


Modula-2 Data Types

The compiler supports all standard Modula-2 types. This implementation also adds several new types to take advantage of the 68000 processor. The following is a description of all the types supported by the compiler, excluding those defined in the pseudo-module SYSTEM, those are described in the previous section about the module SYSTEM. The descriptions associated with each type, only describe information relevant to this implementation, for a full explanation of a particular type refer to a textbook on Modula-2.

set	A set may contain up to 32 elements in the range 0 to 31. The range specified determines the number of bytes needed to represent a set, since each element is represented by one bit. SET OF [0..7] - requires one byte SET OF [0..15] - requires two bytes SET OF [0..31] - requires four bytes
enumeration	An enumeration type may be defined with up to 256 possible constants. An enumeration always occupies one byte.
array	The total size of an array must not exceed 32K bytes. The size of the array is rounded up to an even number of bytes.
record	The size of a record is always rounded up to an even number of bytes.
procedure	The procedure type is a pointer to the executable code of a procedure. The size of this type is the same as for a pointer type, four bytes of storage.
pointer	The pointer type points to an object of a specified type. The size of the pointer type is always four bytes.
subrange	The bounds of a subrange type cannot be larger than the range -32768 to 32767. The size of this type is two bytes.
BOOLEAN	A variable of type BOOLEAN will always contain FALSE(0) or TRUE(1), it is improper to represent TRUE by any value other than one. The size of a this type is one byte.
BITSET	A form of a set defined as a SET OF [0..15]. The type occupies two bytes of storage.
CARDINAL	Represents the range of values between 0 and 65535. The CARDINAL type is assignment compatible with INTEGER, LONGINT and LONGCARD. The assignment to INTEGER does not cause any change in the bit pattern. An assignment to LONGINT or LONGCARD causes the value to be zero extended. The type occupies two bytes of storage.

CHAR	Represents the range of character values between 0 and 255. The type occupies one byte of storage.
INTEGER	Represents the range of values between -32,768 and 32,767. The number is stored in two's complement form. The high-bit holds the sign of the number. The INTEGER type is assignment compatible with CARDINAL, LONGINT and LONGCARD. An assignment to CARDINAL causes no change in bit pattern. The assignment to LONGINT or LONGCARD is performed using sign extension. The type occupies two bytes of storage.
LONGCARD	Represents the range of values between 0 and 4,294,967,295. A LONGCARD is assignment compatible with INTEGER, CARDINAL and LONGINT. An assignment to INTEGER or CARDINAL is performed by truncating the upper 16-bits of the bit pattern. An assignment to LONGINT causes no change in the bit pattern. The type occupies four bytes of storage.
LONGINT	Represents the range of values between -2,147,483,648 and 2,147,483,647. A LONGINT is assignment compatible with INTEGER, CARDINAL and LONGCARD. The assignment to INTEGER and CARDINAL is performed by truncating the upper 16-bits of the bit pattern. An assignment to LONGCARD causes no change in the bit pattern. The type occupies four bytes of storage.
PROC	Procedure type compatible with procedures with no parameters and no return parameter. Occupies four bytes of storage.
REAL	The Motorola Fast Floating Point format is used to represent REAL numbers. The range of values which can be represented by this format is between -9.22337177E+18 and 9.22337177E+18. The REAL type occupies four bytes of storage.

Long Literal Constants

When specifying literals of type LONGINT or LONGCARD the number must be followed by the character "D", meaning double or long. For example:

1D	- LONGINT or LONGCARD
-1D	- LONGINT
131072D	- LONGINT or LONGCARD
2147483647D	- LONGINT or LONGCARD
1	- INTEGER or CARDINAL
-1	- INTEGER
-32000	- INTEGER

Identifiers

The compiler accepts identifiers up to 255 characters long. All characters are significant. Modula-2 specifies that identifiers must consist of only alphanumeric characters, this implementation also allows the underscore character "_" to be used in identifiers. The underscore character is not accepted by most other implementations of Modula-2, so portability must be considered before using this feature of the compiler.

Size Limitations

This section describes the code and data limitations of the Modula-2 compiler. The specified limitations apply only to an individual module. Since most programs consist of more than one module, the specified limitations will rarely be encountered. If a limitation is encountered it can always be overcome by splitting a module into two or more smaller modules. It is very important to understand that the final executable file generated by the linker has no limitations on either code or data, except the memory of the target system which will be used to execute the program.

Name	Maximum Size/Module
Global Variables, Data	32766
Statements, Code	32766
Strings, Data	32766

String Literals

This implementation of Modula-2 provides a much more flexible facility for specifying string literals than is available in most other implementations. Therefore, before using this very useful facility, portability considerations should be taken into account. The following table describes the special commands which may be imbedded into a string. Each command begins with a "\" character. These commands are almost identical to the commands available in the "C" language.

Command	Description
\[LF]	The string continues on the next line.
\n	Put line feed (LF) character into string.
\t	Put horizontal tab (HT) character into string.
\b	Put backspace (BS) character into string.
\r	Put return (CR) character into string.
\f	Put form feed (FF) character into string.
\v	Put vertical tab (VT) character into string.
\x[n][n]	Put the character represented by a hexadecimal code.
\[n][n][n]	Put the character represented by an octal code.
\<AnyChar>	Place the character after the "\" into string.

The following examples demonstrate how the various special commands are translated by the compiler:

"Hello World!\n"	Hello World!<LF>
"\fThe End\\\r"	<FF>The End<CR>
"\x32\x35\t\x30\x31\n"	25<HT>01<LF>
"\101 TO \132"	A TO Z
"\\"	\

When translating a program from a different Modula-2 compiler, the source may have used the character constant "\". This will not compile because the \ would cause the next character " to be placed into the string. To fix the problem replace the "\" with "\\" or use the ASCII value 134C.

Modula-2 Software Construction Set Reference Guide

FORWARD Procedure

To facilitate referencing the name of a procedure identifier before it is defined, a procedure should be declared as FORWARD. The following example demonstrates the use of a FORWARD procedure:

```
PROCEDURE GetNode(): NodePtr; FORWARD;
```

```
PROCEDURE Tree;  
VAR n : NodePtr;  
BEGIN
```

```
  ...  
  n := GetNode();  
  ...  
END Tree.
```

```
PROCEDURE GetNode(): NodePtr;  
BEGIN
```

```
  ...  
END GetNode;
```

CODE Procedure

The CODE procedure is a way of inserting a short instruction sequence into the body of a procedure or module. A CODE procedure does not have a body part like a normal procedure. Its body is represented by a 16-bit code, this code will be inserted into the instruction stream wherever this CODE procedure is called. A CODE procedure can be specified as having arguments, in this case the arguments will be evaluated as usual and placed on top of the stack. A simple example of its use is contained in the following code fragment:

```
PROCEDURE BreakPt; CODE 04AFCH; (* ILLEGAL instruction *)
```

```
PROCEDURE Buggy;  
BEGIN
```

```
  a := b;  
  c := d;  
  BreakPt; (* Causes a break point to be inserted into code *)  
  x := y;  
END Buggy.
```

Run Time Error Checking

Certain types of errors cannot be detected at compile time, therefore, the compiler can be instructed to perform additional error checking at run time. The types of run-time error checking supported are arithmetic overflow, array index and subrange bounds checking. For information on how to enable or disable run time error checking refer to the sections on compiler switches and command line options.

Arithmetic overflow checking applies to operations involving the signed types INTEGER or LONGINT. An arithmetic overflow error occurs when an operation involving two INTEGERS causes the result to exceed the minimum or maximum value which can be represented by the type INTEGER. Arithmetic overflow checking is implemented by using the MC68000 TRAPV instruction.

Array bounds checking causes the index into an array to be checked against the bounds of the array. If the index exceeds the bounds of the array a run time error occurs. Array bounds checking is implemented using the MC68000 CHK instruction.

Subrange bounds checking causes the value being assigned to a variable of a subrange type to be checked for being within the bounds of the subrange type. Subrange bounds checking is implemented using the MC68000 CHK instruction.

The run-time error checking options of the compiler should not be enabled unless the program imports the module RunTimeErrors or the program will be used with a source level debugger or a symbolic debugger. If a run-time error occurs and one of the above tools is not used the Amiga will put up a "Software Error" requester and suspend the program.

Register Usage

This section deals with the run-time register organization of Modula-2 programs. Registers designated as scratch do not need to be preserved by procedures, this includes registers D0 and D1.

- D0 - Lower 32-bits of function call result
- D1 - Upper 32-bits of function call result(64-bit results only)
- D2 - scratch register
- D3 - scratch register
- D4 - scratch register
- D5 - scratch register
- D6 - scratch register
- D7 - scratch register

- A0 - scratch register
- A1 - scratch register
- A2 - scratch register
- A3 - scratch register
- A4 - module static variables base pointer
- A5 - stack frame pointer
- A6 - scratch register (used in Amiga library calls)
- A7 - top of stack pointer

Short And Long Addressing

The compiler supports two methods of accessing static variables within a module. The first method is most efficient when a module has a lot of references to static variables, the static variables are referred to by 16-bit register A4 relative addressing. This requires only 16-bits to encode the address of a variable, this method is both space and time efficient. The overhead in using the register relative addressing is that the A4 register has to be preserved and loaded on entry into a procedure at scope level 0. The second method of referencing static variables is using 32-bit absolute addressing, this method requires both more space and time for each reference. However it does not require the register A4 to be loaded.

For purposes of symbolic debugging it may be useful to use the 32-bit absolute addressing. By using 32-bit absolute addressing the debugger will be able to display symbols for the variables being referenced, most debuggers cannot determine the symbol when the 16-bit register A4 relative addressing is used.

Register Usage

This section deals with the run-time register organization of Modula-2 programs. Registers designated as scratch do not need to be preserved by procedures, this includes registers D0 and D1.

D0	- Lower 32-bits of function call result
D1	- Upper 32-bits of function call result (32-bit results only)
D2	- scratch register
D3	- scratch register
D4	- scratch register
D5	- scratch register
D6	- scratch register
D7	- scratch register
A0	- scratch register
A1	- scratch register
A2	- scratch register
A3	- scratch register
A4	- module static variables base pointer
A5	- stack frame pointer
A6	- scratch register (used in library calls)
A7	- top of stack pointer

Chapter 5	Modula-2 Linker.....	3
	Running the Linker.....	3
	The Version Control System.....	4
	Symbolic Debugging.....	5
	Run-Time Support Library.....	5
	Executable File Structure.....	5
	Linker Limitations.....	6

Chapter 2: Module-2 Linker	2
Running the Linker	3
The Version Control System	4
Symbolic Debugging	5
Run-Time Support Library	5
Executable File Structure	5
Linker Limitations	6

This section deals with the linker program included in this package. The purpose of the linker is to combine object modules into one executable file acceptable to the AmigaDOS operating system.

The linking process for a Modula-2 program is much simpler than for many earlier languages such as "C" or "PASCAL". To link a program it is only necessary to know the name of the "main" module. Any modules required to link the "main" module are linked in automatically without being explicitly specified.

Running the Linker

The name of the linker program is "M2Lk". The syntax for running the linker is as follows:

```
M2Lk [Options] <MainModule> [MainModule] ...
```

The name of the module to be linked must be specified on the command line. The name of the main module should correspond with the name of the object module, excluding the extension ".OBM". If more than one "MainModule" is listed each one will be linked in the order they are listed on the command line. While linking more than one "MainModule" it is possible to interrupt the linking process by pressing CTRL-C. The linking process will stop as soon as the current "MainModule" has been linked.

The linker will display names of the object files being linked in. Each name will be preceded by the symbol "<-" to indicate input. If the linker has been instructed to generate symbols, the names of symbol or reference files that have been successfully read will be displayed preceded by the symbol "Sym" to indicate symbolic information. After the linking is complete the linker will display the name of the executable output file preceded by a "->" to indicate output. The size in bytes of each section of the executable file will be displayed on the next line. The explanation for the size information displayed by the linker is as follows:

```
Code   - executable code, statements
UData  - uninitialized data, static variables
IData  - initialized data, string literals
Init   - size of module initialization jump table
Total  - total size of the executable file
```

Before, after or between the names of the modules to be linked, one or more command line options may be specified. A command line option begins with the character "-" followed by a letter specifying the option. An option may be a toggle switch or it may be followed by an additional argument. The following is a list of the available linker options and the purpose of each option:

* **-s <DirectoryPath>**

Specifies the name of a directory to be searched for object files(.OBM), symbol files(.SBM), or reference files(.RFM). The option may be used more than once to specify additional directories to be searched. By default the "M2L:" directory is searched automatically, the directory should contain the system library modules which all programs will need.

Example: M2Lk Draw -s RAM:

* **-o <DirectoryPath>**

Specifies the name of the directory to which files generated by the linker should be written.

Example: M2Lk Draw -o RAM:

* **-d**

Instructs the linker to add symbols to the executable file. This allows a symbolic debugger to display the names of variables and procedures while debugging.

Example: M2Lk Draw -d

* **-a**

This option is a modifier for the "-d" option. This option causes the linker to prefix each symbol with the name of the module in which the symbol is defined. For example if the symbol "WriteLn" is defined in module "InOut" the symbol stored will be "InOut_WriteLn". This option may be useful when symbols with the same name are defined in different modules.

Example: M2Lk Draw -d -a

* **-g**

This option instructs the linker to add a special debug block to the executable file. This option is used when the program may be executed from the source level debugger. The resulting executable file will be a bit larger than normal, but this does not increase the amount of memory used by the program at run-time.

Example: M2Lk Draw -g

The Version Control System

The compiler section of this manual described the version control system used by this implementation of Modula-2. The linker is an integral part of this version control system. The linker will not generate an executable file unless all of the modules that are linked together have compatible version control keys.

The linker will display an error message if it finds a module with an invalid version control key. To rectify the problem, one or more modules will need to be recompiled. The modules that may need to be recompiled are dependent on the module hierarchy of the program being linked.

Symbolic Debugging

Programs written in Modula-2 may be debugged using an assembly language debugger. If the debugger is capable of displaying symbols, then it is called a symbolic debugger. Using a symbolic debugger will greatly enhance the ease with which debugging can be performed.

If the debugger supports symbols, the linker can be instructed to output the symbols to the executable file. The debugger will read in the symbols and the user will be able to refer to variables and procedures by name rather than by address, this greatly simplifies the debugging process. The linker command line options "-a" and "-d" are used to instruct the linker to add symbols to the executable file.

The linker will search for a reference file (extension .RFM). The reference file is generated by the compiler when the "-g" option is specified on the compiler command line. Refer to the chapter on the compiler for further information about reference files. If the reference file is not found, the linker will search for a symbol file (extension .SBM). If the symbol file is not found no symbols will be added for that specific module. If a symbol or reference file is found, the linker will read in the names of the procedures and the names of the global variables and place the symbols into the executable file. This action is performed for each module linked in to the program.

Run-Time Support Library

To provide run time support for a Modula-2 program, the linker implicitly links in a module called **System**. The contents of this module are explained in another section in this manual.

Executable File Structure

The executable file generated by the linker has the following format. The code and data sections of each module occupy one hunk in the executable file. The first hunk in each executable file contains a jump table to the module body of each module. The last jump in the table is a jump to the module body of the "main" module. The linker places each module in a separate hunk allowing a large program to be loaded into a system where the memory has been fragmented into many small pieces.

It should also be noted that the size of the executable file is larger than the amount of memory actually occupied by the program once loaded into memory. This is because the file contains relocation information necessary to load the program, this information is automatically discarded when the program is loaded. When symbols are added to an executable file, the size of the file will also increase. However when the program is executed outside of a debugger it will not require any additional memory to run, the AmigaDOS loader does not load the symbols into memory.

Linker Limitations

The linker has no size limitations on either the number of modules which can be linked nor on the size of the final executable file. The amount of memory in the system does not limit the size of the program which may be linked, because only one object module is held in memory at any one time.

The linker will search for a reference file (extension .RPM). The reference file is generated by the compiler when the "-g" option is specified on the compiler command line. Refer to the chapter on the compiler for further information about reference files. If the reference file is not found, the linker will search for a symbol file (extension .SYM). If the symbol file is not found no symbols will be added for that specific module. If a symbol or reference file is found, the linker will read in the names of the procedures and the names of the global variables and place the symbols into the executable file. This action is performed for each module linked in to the program.

Run-Time Support Library

To provide run time support for a Modula-2 program, the linker implicitly links in a module called System. The contents of this module are explained in another section in this manual.

Executable File Structure

The executable file generated by the linker has the following format. The code and data sections of each module occupy one link in the executable file. The first link in each executable file contains a jump table to the module body of each module. The last link in the file is a jump to the module body of the "main" module. The linker places each module in a separate link allowing a large program to be loaded into a system where the memory has been fragmented into many small pieces.

Chapter 6	EMACS Editor	3
Running the Editor		3
The Editing Environment		3
EMACS Terminology		4
Keyboard Commands		6
Menu Commands		8
Mouse Commands		8
Overwrite Mode		10
Fill Mode		10
Delayed Prompts		10
Command Prefix Argument		11
Rebinding Keys		11
Expression Evaluation		12
Echo Line Prompt		13
Auto Completion of Names		13
Executing Commands by Name		14
Keyboard Macro		14
The EMACS Program Window		15
The File Requester		15
The Current Directory		17
Exact Filename Mode		17
The Modula-2 Compiler		18
The Modula-2 Linker		20
Running Modula-2 Programs		21
Command Descriptions by Topic		22
Cursor Commands		22
Window Commands		23
Buffer Commands		24
File Commands		25
Copy/Delete/Kill Commands		26
Search/Replace Commands		28
Case Commands		29
Key Binding Commands		30
Macro Commands		30
Fill Mode Commands		30
Expression Commands		31
Miscellaneous Commands		31
Mouse Commands		34
Display Setup Commands		35
Modula-2 Compiler Commands		36
Modula-2 Linker Commands		37
Modula-2 Run Commands		38

Chapter 6: EMACS Editor	3
Running the Editor	3
The Editing Environment	3
EMACS Terminology	4
Keyboard Commands	4
Menu Commands	5
Mouse Commands	5
Overwrite Mode	10
Fill Mode	10
Delayed Prompts	10
Command Prefix Argument	11
Redefining Keys	11
Expression Evaluation	12
Echo Line Prompt	13
Auto Completion of Names	13
Executing Commands by Name	14
Keyboard Macros	14
The EMACS Program Window	15
The File Reducator	15
The Current Directory	17
Exact Filename Mode	17
The Modula-2 Compiler	18
The Modula-2 Linker	20
Running Modula-2 Programs	21
Command Descriptions by Topic	22
Cursor Commands	22
Window Commands	23
Buffer Commands	24
File Commands	25
Copy/Deface/Kill Commands	26
Search/Replace Commands	28
Case Commands	29
Key Binding Commands	30
Macro Commands	30
Fill Mode Commands	30
Expression Commands	31
Miscellaneous Commands	31
Mouse Commands	34
Display Setup Commands	35
Modula-2 Compiler Commands	36
Modula-2 Linker Commands	37
Modula-2 Run Commands	38

The editor included in this package is a full-screen display editor modeled after EMACS, the editor created by Richard M. Stallman at the MIT Artificial Intelligence Laboratory.

In addition to providing many commands to support editing of Modula-2 source files, EMACS has an integrated Modula-2 Compiler and Linker. By integrating these components together this package provides an extremely productive environment for developing programs in the language Modula-2.

Running the Editor

The name of the editor program is "M2Ed" and the command line syntax for the program is as follows:

```
M2Ed [FileName] [FileName] ...
```

The [FileName] specifies the name of a file to be loaded into a buffer. The files specified on the command line will be loaded into buffers upon entry into the editor, the editor window will display the buffer of the last file loaded. If no filenames are listed on the command line the editor displays a default buffer called "*Scratch*" in the window.

The Editing Environment

This subsection describes the basics that are needed to understand the EMACS editing environment.

The EMACS window

This is the window the EMACS program uses for all input and output activities. This window is opened on the WorkBench and it may be resized, moved or rearranged as any other Amiga window. Most references to a window in this document are not referring to the EMACS window, rather they are referring to the sub-windows created within the EMACS window. These sub-windows are used to simultaneously display the contents of one or more edit buffers.

The edit buffer

This is a temporary storage area for lines of text being edited. In this document the edit buffer is usually referred to as the "buffer" or "current buffer", the latter term referring to the buffer currently being edited.

When in EMACS, there is at any particular time just one particular buffer where editing occurs, the current buffer. Editing consists of adding or deleting characters at various places in the buffer.

At any particular time in the editing process the buffer may be empty, or it may contain a number of lines of text typed in by the user or loaded from a disk file.

While in EMACS you may create as many buffers as necessary limited only by the amount of memory in the system. Each buffer may be manipulated

individually, with no effect on any other buffer. A buffer may be erased, loaded with text from a file, written to a specified file or many other possible operations may be performed on a buffer.

A text file

This is a file containing ASCII characters which may be read into a buffer for editing purposes. Editing a file using EMACS consists of loading the file into a buffer, making changes to the buffer and then writing the buffer back to the file from which it was read or another file if desired.

Since editing of the buffer does not change the original file stored on disk it is important to save changes prior to exiting EMACS. EMACS will issue a warning if you attempt to leave the program while one or more buffers have not been saved to their corresponding files. It is a good idea to save your changes to the file from time to time while you are working to minimize loss of data which may result if a power loss is experienced during the editing session.

The edit window

This is a view into the contents of a specific edit buffer. Multiple edit windows may be setup to allow viewing of several different buffers or different parts of the same buffer simultaneously.

The number of edit windows which can be open at the same time depends on the number of lines on the display. Each edit window is at least one text line in height. The edit windows are separated from each other by a solid line containing information relating to the contents of that window.

EMACS Terminology

The following terms are used throughout this document to describe functions available in EMACS.

BUFFER

A temporary storage area used for editing the contents of files. The buffer in which the cursor is currently located is called the current buffer.

POINT

The current cursor position in the buffer is referred to as the "point". Deletion and insertion operations take place at the point. When moving from one buffer to another the position of the point is remembered so that when you return to that buffer the cursor will return to the same position.

MARK

The mark is similar to the point except that it is invisible. To set the mark to a specific location in the buffer, the point is moved to the desired location and a set-mark-command (Ctrl-@) is issued. The mouse may be used to set the mark by double-clicking the left button on the desired location. The mark is also set implicitly by other EMACS commands.

REGION

The text between the point and the mark is called the region. EMACS allows

operations to be performed on a region, such as copying a region or killing a region.

PARAGRAPH

EMACS has been given some primitive word processing capabilities to format text. In EMACS, a paragraph is considered to be a block of text separated above and below by a blank line or by a line which begins with a blank character. The first line of a paragraph may begin with one or more white spaces.

KILL

Commands that specify "kill" rather than "delete" place the characters which are deleted into a kill buffer. The text in a kill buffer can be inserted into a normal buffer using the yank command.

YANK

Insert the contents of the kill buffer into the current buffer at the point. If the kill buffer is empty then nothing happens.

WINDOW

A number of horizontal lines of text which provide a view into a buffer. Different parts of one buffer or multiple buffers may be viewed simultaneously by opening several windows. Windows are created by splitting an existing window into two windows. After finishing with a window it may be collapsed down into another window.

MODE LINE

This is a solid line which appears below each window on the display. This line contains the following information:

- * Window Gadgets - Three gadget which operate on the window are displayed.
 - [*] = delete window
 - [v] = enlarge window
 - [^] = shrink window
- * Status Indicator - If the characters '***' appear on the line then changes have been made to the buffer, but the changes have not been saved to a file.
- * Modes - Currently enabled edit modes are displayed in parenthesis.
- * Buffer Name - The name associated with the buffer, followed by a colon.
- * File Path - The absolute file path associated with the given buffer. If the buffer has not been read from or written to any file, "null pathname" is displayed to indicate that no file is associated with the buffer.

ECHO LINE

This is the line at the bottom of the display which is used by EMACS to communicate with the user. EMACS will display messages on the echo line as well as prompt the user to enter additional information at various points in the editing process.

KEY BINDING

The association of a user typed key combination with a command or function

available in EMACS. Editor functions are not "hard wired" to the key combinations and may be completely redefined to the user's preferences.

Keyboard Commands

EMACS based editors have been around for a long time and have had the benefit of input from thousands of experienced and novice users. This experience has been used to provide a very efficient and easy to learn editing environment. Since EMACS originated on systems which did not have alternative input devices such as the mouse, it has been designed to take maximum advantage of the keyboard. The default key bindings used in EMACS are a result of years of refinement and in most cases the key binding is logical because of its position on the keyboard or because the label on the key is related to the function it performs. However, to provide the maximum user convenience all key combinations are completely reconfigurable by the user. Those users who wish to avoid using the keyboard may use the mouse and the pull-down menus for many of the commands available from the keyboard. However, if speed of editing is an important consideration, it will eventually be necessary to master the keyboard.

Most EMACS key bindings are invoked through one of the following methods:

- * Ctrl-<Char> = To invoke a command, press and hold the key labelled <CTRL> followed by the specified character.
- * Esc-<Char> = To invoke a command, press the key labeled <ESC>, release the key, then type the specified character.

The key labelled <CTRL> is called the control key. While the key labeled <ESC> is known as the escape key or meta key.

The following are examples of valid EMACS key combinations:

- Ctrl-p = Press and hold the control key and type "p".
- Esc-a = Type the escape key and then type "a".
- Ctrl-x f = Press and hold the control key and type "x", then release the control key and type "f".

A very important command to remember is the abort command, this command is "hard wired" to the Ctrl-g key combination. This command may be invoked from anywhere in EMACS, including a partially entered key combination or a prompt. This command is very useful in many situations. For example if the first part of a key combination has been typed, but the user does not remember the complete key combination, the command may be aborted rather than invoking the wrong command which may have an undesirable effect.

A very useful command is available for checking the function which will be executed when a given key combination is pressed. This command is invoked via the <HELP> key. After pressing the <HELP> key a prompt will appear, type any valid key combination and the function associated with that key combination will be displayed.

At some point it will probably be necessary to exit EMACS and return to the

CLI, the command to exit is Ctrl-x Ctrl-c. Before exiting EMACS will check if any buffers have been modified, but not saved to disk. If this is the case, EMACS will prompt the user to save the buffers before exiting.

In addition to the normal keyboard keys the arrow keys, function keys and other non alpha-numeric keys have special meaning when using the EMACS editor.

The arrow keys, both shifted and unshifted, have been assigned EMACS cursor movement commands. The following table describes the default bindings for the arrow keys:

Key	Shift	Command
Left		backward-char
Left	*	backward-word
Right		forward-char
Right	*	forward-word
Up		previous-line
Up	*	backward-paragraph
Down		next-line
Down	*	forward-paragraph

The function keys, both shifted and unshifted, have been assigned EMACS commands, most of these commands relate to the Modula-2 Compiler and Linker. The following table describes the default bindings for the function keys:

Key	Shift	Command
F1		m2-err-next-error
F1	*	m2-err-current-error
F2		m2-comp-current-buffer
F2	*	m2-comp-buffer
F3		m2-link-main
F3	*	m2-link-module
F4		m2-run-main
F4	*	m2-run-module
F5		m2-comp-file
F5	*	free-mem-avail
F6		set-m2-main-module
F6	*	set-m2-link-out-dir
F7		set-m2-obj-dir
F7	*	set-m2-link-gen-sym
F8		m2-comp-load
F8	*	m2-comp-unload
F9		find-file
F9	*	find-file-other-window
F10		save-buffer
F10	*	write-file

Menu Commands

EMACS provides menus for many of the available commands. To examine the menus, press and hold the right mouse button. At the top of the screen will appear a menu bar with the available menus listed by subject. Move the pointer to one of the menu names and the associated menu will appear below the menu name. Each item in the menu represents an EMACS command along with its default keyboard binding. To invoke the command move the pointer to the desired command and release the button. Multiple menu commands may be specified without releasing the right mouse button by positioning on the desired item and clicking on the left mouse button, each command will then be executed in the order received. The menus can also serve as a quick reference to find the default key binding for a given command.

Mouse Commands

EMACS has been designed so that it may be used without ever touching the mouse throughout the entire editing session. However, to accommodate users who would like to use the mouse it is possible to execute 24 commands from using mouse.

The mouse commands are divided into three groups. Each group is associated with a specific part of the EMACS display. The three parts of the display are the text area, the mode line and the echo line. The display component on which the pointer associated with the mouse is located determines the command types which may be executed. All mouse commands use the left mouse button, all further references to clicking the mouse imply the usage of the left mouse button.

The mouse commands in the text area group effect the text in the buffer being displayed. When the mouse is clicked in the text area of a window the point associated with the buffer displayed in the window is moved to the character under the mouse pointer. Then the command bound to the mouse button is executed. The function performed by the command is the same as when the command without the "mouse-" prefix is executed. The mouse may also be used for setting the "mark", this is done by double-clicking on the desired location.

If the mouse is clicked in the mode line of a window, a command from the mode line group is executed for the specified window. The point for the buffer associated with that window is set to its previous position and the command bound to the mouse button is executed.

The echo line of the display is not related to any specific window. Therefore, clicking the mouse while pointing to the echo line will execute a mouse command which has a global effect on EMACS.

Each mouse command group contains eight commands which may be executed, to select a specific command the mouse is clicked while some combination of the Ctrl, Alt and/or Shift keys is pressed. The following three tables show the

commands possible and the key combinations required to execute the commands. A "*" character in a column indicates that the key has to be held down while clicking the mouse.

Text Area Commands

Ctrl	Alt	Shift	Command
		*	amiga-mouse
		*	mouse-recenter
	*	*	mouse-kill-word
	*	*	mouse-kill-line
*			mouse-delete-char
*		*	mouse-just-one-space
*	*		mouse-kill-region
*	*	*	mouse-yank

Mode Line Commands

Ctrl	Alt	Shift	Command
		*	mouse-scroll-up
		*	mouse-scroll-down
	*	*	mouse-split-window-vertically
	*	*	mouse-delete-window
*		*	mouse-beginning-of-buffer
*		*	mouse-end-of-buffer
*	*	*	mouse-enlarge-window
*	*	*	mouse-shrink-window

Echo Line Commands

Ctrl	Alt	Shift	Command
		*	switch-to-buffer
		*	kill-buffer
	*	*	describe-key-briefly
	*	*	describe-bindings
*		*	suspend-emacs
*	*	*	save-buffers-kill-emacs
*	*	*	list-buffers
*	*	*	toggle-window-hack

Modula-2 Software Construction Set Reference Guide

The commands executed when the key combination and a mouse click is performed may be reassigned to any emacs command using the `global-set-key` command. When the `global-set-key` command prompts for the key to be reassigned, perform the key combination and the mouse click which is to be reassigned, EMACS will take this action as if it were a normal key sequence.

Overwrite Mode

Normally when the user types a character EMACS inserts the character into the current buffer and moves the cursor forward, this called the insert mode. An alternate mode of editing is available called overwrite mode. The command `"overwrite-mode"` is used to toggle the mode on or off, when the mode is enabled the mode line will display the word `"over"`. In overwrite mode when a character is typed it overwrites the character at the cursor position rather than inserting the character. If the cursor is at the end of line or no more lines exist then EMACS inserts any characters typed just as in insert mode. In addition, when typing `<RETURN>` (which executes the command `"insert-newline"`) EMACS will move the cursor to the beginning of the next line rather than inserting a new blank line as it would in insert mode, however, if the cursor is positioned at the end of the file a new line will be inserted. When using the `"auto-indent"` mode the binding of `Ctrl-J` and `<RETURN>` keys is reversed, in this case even while overwrite mode is active the `<RETURN>` key still performs a newline and indent operation.

Fill Mode

A limited word processing capability is available in EMACS called fill mode. The command `"auto-fill-mode"` is used to toggle the mode on or off. When the auto fill mode is enabled the mode line will display the word `"fill"`. When in fill mode the `<RETURN>` key does not have to be typed at the end of each line, EMACS will automatically wrap words to the next line when the left margin is crossed.

The `set-fill-column` function activated by `Ctrl-x f` may be used to change the position of the left margin, also known as the fill column. To set the fill column move the cursor to the column to be the left margin and invoke the `set-fill-column` function.

The `fill-paragraph` function activated by `Esc-q` may be used at any time to reformat a paragraph, EMACS does not have to be in fill mode in order to use this function. This function may be used after a new fill column has been set or after a paragraph has been changed to reformat the paragraph.

Delayed Prompts

If the user pauses while typing a command which consists of more than one keystroke, EMACS will display the keystrokes entered on the echo line. This is done to remind the user what has been keyed in so far.

Command Prefix Argument

The command prefix allows a parameter to be passed to a command. This argument usually specifies a repeat count for the command, allowing a given command to be executed a specified number of times. In some cases this argument is used to cause the command to behave differently than it would without any argument. The command prefix is introduced by typing Ctrl-U followed by an optional minus sign if a negative number is desired, followed by a decimal number up to 32,767. To change the number the <BackSpace> key may be used to delete the number digit by digit. If a number is not entered, the default argument of (4) is assumed. The number is usually entered without any feedback on the display, however, if the user pauses, the delayed prompting will cause the number to appear on the echo line.

Rebinding Keys

A very important feature of EMACS is the ability to completely redefine the keyboard. Each function in EMACS has a name which identifies the function. The user can associate a valid key combination with a function, thereafter, whenever the key combination is typed the associated function will be executed.

The function `global-set-key` is used to associate a function with a key combination. The function prompts for the key combination to be defined, at this prompt type the exact key combination, for example Ctrl-x h. If a valid key combination was typed, the function then prompts for the name of the function to be associated to that key combination, for example `find-file`. Thereafter, whenever this key combination (Ctrl-x h) is issued the function (`find-file`) will be invoked. If the `global-set-key` command is used on an already defined key combination, the new definition replaces the old definition.

The function `global-unset-key` is used to disassociate a function from a key combination. The function prompts for the key combination to be deleted. Thereafter, typing the key combination will cause nothing to happen.

The function `describe-key-briefly` may be used to determine the function associated with a given key combination. This function is bound to the <HELP> key by default and is very useful for beginners trying to learn the key combinations.

The function `describe-bindings` creates a list of all the currently active key bindings. This list is generated into a new buffer called `"*Help*"`. It may be useful to print this buffer on a printer and use the hard copy for reference purposes.

While it is possible to change key bindings manually each time EMACS is started, it is not very convenient. A much more practical approach is to add the key bindings to the default EMACS startup file called `".emacs"`.

Expression Evaluation

This EMACS facility tries to emulate a tiny portion of the LISP language built into EMACS on larger systems. The purpose of this facility is to read a file containing EMACS commands, typically these are commands which rebound keys. Each EMACS command has an associated symbolic name, the command may be bound to any number of possible key combinations at the same time.

Upon startup EMACS looks for a file called ".emacs" in the current directory and, if not found, then in the "s:" directory. If the file is found it is read and the commands in the file are executed.

To execute a file containing EMACS commands any time during the editing process use the load command. The contents of the specified file will be executed as if they were entered by the user.

The format of a command file is one or more lines of text each containing the name of an EMACS function followed by one or two optional arguments. An optional parenthesis may enclose the entire command.

Two types of arguments are allowed in a command file, a string and a key sequence. A key sequence is equivalent to typing the keys on the keyboard to invoke a command, it must be enclosed in double quotes. A string is what would be typed on the echo-line in response to a prompt from a command, a string may be preceded by an optional single quote.

The following escape sequences may appear inside a key sequence to introduce non-printing characters into the command file:

\^	- next character specifies a control character
\e	- next character specifies an escape character
\n	- newline character (Line Feed)
\t	- tab character
\r	- carriage return character
\f[n][n]	- function/arrow/special keys
	01-10 = F1..F10
	11-20 = Shift F1..F10
	21 = Up Arrow
	22 = Shift Up Arrow
	23 = Down Arrow
	24 = Shift Down Arrow
	25 = Left Arrow
	26 = Shift Left Arrow
	27 = Right Arrow
	28 = Shift Right Arrow
	29 = HELP
\\	- backslash character
\	- next character is inserted "as-is"

The following example rebinds the key combination Esc-o to the command open-line:

```
(global-set-key "\eo" 'open-line)

global-set-key - command to change the binding of a key
"\eo"          - the key to change is Esc-o
'open-line     - change binding to open-line command
```

This example rebinds the key combination Ctrl-x c to the command m2-comp-buffer:

```
(global-set-key "\^xc" 'm2-comp-buffer)

global-set-key - command to change the binding of a key
"\^xc"        - the key to change is Ctrl-x c
'm2-comp-buffer - change bindings to m2-comp-buffer command
```

This example sets a new maximum value for the compiler error count:

```
(set-m2-err-count '50)

set-m2-err-count - command to change error count
'50              - set new error count to fifty
```

Other Examples:

```
(file-req-mode)
(global-set-key "\f1" 'm2-comp-load)
(set-mode-background 1)
```

Echo Line Prompt

Many of the commands in EMACS display a prompt on the echo line of the display. At this point EMACS is expecting the user to type a string in response. The following special keyboard commands are available during the input of a string:

```
Ctrl-g      - abort current function
Ctrl-q      - quote the next character (followed by control character)
Ctrl-u      - erase the entire string
Ctrl-x      - same as Ctrl-u
<BackSpace> - erase the previous character typed
<RETURN>    - terminate string and execute function
```

Auto Completion of Names

EMACS supports a facility called auto completion when entering the name of a command or the name of a buffer. This means that when EMACS prompts for the

name of a buffer or the name of an EMACS function, you can type only the number of characters which are necessary to distinguish the name from the possible list of names. EMACS will automatically complete the name.

The following keys have a special meaning while the auto completion facility is active:

- <SPACE> - complete the current word
- <TAB> - complete the entire name
- <RETURN> - complete the entire name and execute function

If EMACS cannot determine which name is being entered it will display the message "[Ambiguous]". To continue, enter one or more additional characters which will help EMACS to determine which name is being entered.

Executing Commands by Name

Any commands in EMACS can be executed by explicitly typing its name. To do this the command `execute-extended-command` (Esc-X) is specified. The command will prompt for the name of the function to be executed. Type out the name of the function you wish to execute and press <RETURN> the command will be executed. This facility is especially useful for commands which are not bound to any keys.

Keyboard Macro

A simple macro facility is available in EMACS. Currently only one macro may be defined at any one time. The macro feature is primarily useful to perform an editing operation consisting of multiple steps one or more times.

To begin a macro definition the function `start-kbd-macro` is used, normally bound to Ctrl-x (. Thereafter any key or key combination typed is executed as usual, but is also added to the macro. To end the macro definition the function `end-kbd-macro` is invoked, normally bound to Ctrl-x). After a macro definition has been successfully defined it may be executed using the function `call-last-kbd-macro`, normally bound to Ctrl-x e.

To have the macro execute more than once, precede the `call-last-kbd-macro` function with a numeric argument.

Some care should be taken when defining a macro, if one of the functions you are adding to the macro fails, the macro definition will terminate immediately. In addition, when executing a macro, if a command being executed fails, the macro execution will terminate immediately.

To increase the speed of macro execution, the display is not updated while the macro is executing. When macro execution has completed the display will be updated.

The following example defines a simple macro sometimes useful during program development. The commands invoked by this macro are as follows:

- * m2-comp-current-buffer
- * m2-link-main
- * m2-run-main

To define this macro type the following key combinations:

[Ctrl-x (] [F2] [F3] [F4] [Ctrl-x)]

Do not type "[" "]" as they are only here to make it easier to read the macro definition. The commands F2,F3,F4 refer to the corresponding function keys.

Executing this macro causes the current buffer to be compiled. If no errors are encountered during compilation, the program is linked. If no errors are encountered during linking the program is executed. This macro provides a very primitive "make" capability.

The EMACS Program Window

When EMACS starts up, it opens an Intuition window. The window occupies the entire display and has the borders turned off to allow a full 80 columns of text. The size of the initial window is determined by the size of the WorkBench screen in either interlaced or non-interlaced mode. The only limitation on the window is the number of lines of text, currently this value is 61 lines, enough to accommodate a PAL Amiga in interlaced mode. To make a smaller window the command toggle-window-hack is used. After issuing the command the borders will become visible and the window can be resized and repositioned. The toggle-window-hack command can then be used to toggle between a full size display window and a smaller window positioned anywhere on the screen.

The File Requester

Whenever EMACS requires the name of a file to be specified two methods of input are available. The first method places a prompt on the echo line and waits for the user to type in a filename followed by <RETURN>. The second method opens a small Intuition window (on top of the EMACS window) containing a directory listing. The desired file may then be selected using the mouse.

The input method used depends on the state of the "file-req-mode". The "file-req-mode" is initially active when EMACS starts up, meaning that filename input takes place through the file requester window. To toggle the input method to the keyboard input mode execute the function "file-req-mode" again.

The contents of the file requester window is as follows:

- * Title Bar - contains the name of the operation to be performed on the selected file. The title bar may be used to move the window to a new position.
- * Close Gadget - Cancels the file requester without making a file selection.
- * Front/Back Gadgets - These two gadgets allow the window to be moved in front of or behind other windows on the WorkBench screen.
- * "Directory" Gadget - contains the relative path of the directory being displayed. If the gadget is empty then the current directory is being displayed.
- * "File Name" Gadget - name of currently selected file, the filename can be typed into this gadget or can be selected by clicking on a filename in the directory display. When the file requester window first opens this gadget is automatically selected.
- * "Pattern" Gadget - AmigaDOS file pattern to use for determining which filenames in a directory to display. By default the pattern is set to "#?" which matches any name.
- * Directory Display - contents of the directory specified by the "Directory" gadget is displayed, only filenames which match the current pattern are shown. File names are shown in white and directory names are shown in orange. All directory names are shown regardless of the current pattern. Clicking on a filename highlights the name, double-clicking on a filename selects that file. Double-clicking on a directory name displays the files in that directory.
- * Scroll Bar - the vertical scroll bar is used to scroll through the directory entries. The size of the scroll bar is proportional to the number of files visible.
- * ACCEPT Gadget - Perform the specified operation on the currently selected file.
- * VOLUME Gadget - Display a list of volumes currently mounted, double-clicking on a volume name will display the root directory of that volume in the directory display.
- * PARENT Gadget - Display the parent of the current directory.
- * CANCEL Gadget - Cancels the file requester without making a file selection.

The "Directory", "File Name" and "Pattern" gadgets are string input gadgets and support the following special commands:

- * Alt-X - clear contents of string input gadget
- * Alt-Q - restore contents to previous contents (undo)
- * Left - move cursor one character left
- * Shift-Left - move cursor to beginning of string
- * Right - move cursor one character right
- * Shift-Right - move cursor to end of string

The "Pattern" gadget is used to specify an AmigaDOS filename pattern to be applied to the filenames in the directory. Only filenames which match the currently specified pattern will be displayed. The entire filename pattern

capability of AmigaDOS may be used. For additional information refer to the AmigaDOS user's guide.

The following special characters may appear in a pattern definition:

?	= matches any single character
%	= matches the null string
#<p>	= matches zero or more occurrences of pattern <p>
<p1><p2>	= matches a sequence of pattern <p1> followed by <p2>
<p1> <p2>	= matches if either pattern <p1> or pattern <p2> match
()	= groups patterns together

Examples of valid patterns which may be specified in the "Pattern" gadget:

#?	= Match any name
#?.def	= Match any name ending in ".def"
#?.def #?.mod	= Match any name ending in ".def" or ".mod"
Simple#?	= Match any name which begin with "Simple"

If the file requester window is displayed, EMACS commands may not be executed, after a file is selected or the file requester is cancelled you may continue editing.

If the file requester window is moved by the user to a new position, the window will appear at the new position on subsequent invocations of the file request window.

The Current Directory

The current directory is the default directory which will be used by EMACS. Initially, the current directory is the directory in which EMACS was started. This may be changed at any time using the "current-dir" function. This function will display the absolute path to the current directory, if the volume is no longer mounted the message "[VOLUME NOT MOUNTED]" is displayed. At the prompt a new current directory may be specified.

EMACS automatically expands all filenames supplied by the user to the full absolute file path. Because of this, when the current directory is changed the path will continue to be valid. An absolute path is one beginning with a device name or volume name.

When the current directory is changed, the change effects the editor, compiler and linker simultaneously. This way, while EMACS may be started in a sub-directory of DF1: switching the current directory to RAM: will cause the compiler to send output files to RAM:.

Exact Filename Mode

When using a command to read data from a file, EMACS associates the filename with the buffer into which the data is read. AmigaDOS does not

distinguish between lower case and upper case names. If a name in a different case than the filename on disk is specified, EMACS will read in the file. However, when writing out the file the new name will be used. To preserve the original case of the filename without having to remember the case when reading files the "exact-fname-mode" can be used. This mode can be toggled using the "exact-fname-mode" function. Initially EMACS begins with "exact-fname-mode" disabled. The "exact-fname-mode" affects all the read and write functions of EMACS.

The Modula-2 Compiler

The Modula-2 compiler has been integrated with EMACS to provide convenient compilation of Modula-2 programs from within the editor. This subsection will describe how the compiler interacts with EMACS. For more information on the compiler itself refer to the compiler chapter of the manual.

Before a program can be compiled from within EMACS, the Modula-2 compiler must be loaded into memory. The compiler is loaded into memory automatically anytime a compiler related command is used. To explicitly load the compiler, invoke the command `m2-comp-load`. The command will prompt for an argument string to be passed to the compiler. The argument string may contain any of the arguments available when using the compiler from the CLI. For details on the arguments allowed refer to the compiler chapter. After typing the argument string press the `<RETURN>` key. If the compiler is loaded automatically then an empty argument string will be passed to the compiler, to pass an argument string the compiler must be loaded explicitly using the `m2-comp-load` command. EMACS will try to load the compiler from disk, the file name of the compiler is "M2". EMACS first tries to load the compiler from the current directory, if it is not found, EMACS searches the AmigaDOS PATH and finally the "C:" directory. If EMACS has any difficulties loading the compiler an error message will be displayed on the echo line. After the compiler is loaded successfully the compiler is passed the specified argument string. If the argument string is invalid an error message will be displayed and the compiler will be unloaded from memory. If the compiler accepts the argument string and there is no other problems, the message "Modula-2 Compiler Loaded." is displayed on the echo line.

After the compiler is successfully loaded it becomes resident. The compiler will remain resident until EMACS is exited or the compiler is explicitly unloaded using the command `m2-comp-unload`.

As long as the compiler is resident it may be used to perform the following operations:

- * Compile the contents of the current buffer
- * Compile the contents of a specified buffer
- * Compile a file on disk

The buffer associated with the window in which the cursor is currently positioned is considered the current buffer. To compile the contents of the buffer the command `m2-comp-current-buffer` is invoked. The compiler will compile the current buffer returning with a message indicating if any errors were found

in the compiled text.

The command `m2-comp-buffer` may be used to specify the name of a buffer to be compiled. This works the same as compiling from the current buffer except that the buffer does not have to be visible on the display to be compiled.

While compiling from a buffer the echo line will be updated to show the current line number being compiled. This update will occur every 32 lines. At the beginning of the buffer where the import lists are usually located the update will be performed every 2 lines, to give a better idea of the progress.

On an Amiga with only 512K of memory it may not be possible to compile a very large program directly from a buffer. In this case the command `m2-comp-file` may be used to compile a file located on disk.

After compilation EMACS may display the message "Symbol Module Generated." for a DEFINITION module or "Object Module Generated. Code:xxxx UData:xxxx IData:xxxx Total:xxxx" for an IMPLEMENTATION module, "xxxx" represents the size of the module in bytes. This means that the compilation was successful and the appropriate output files have been generated. If any errors were found in the source text, the message "Source Compiled, n Errors Found." where n is the number of errors encountered in the source text.

If one or more errors were found in the source, it is now possible to step through each error one at a time. The cursor is moved to the location of each error and the associated error message is displayed in the title bar of the window and on the echo line. The contents of the echo line may change after any command is issued, while the title bar will continue to display the error message.

To begin fixing errors in the source text, use the command `m2-err-next-error` to move the cursor to the position of the first error. If the current window is not displaying the buffer which contains the error, EMACS will force the contents of the current window to display the correct buffer. The cursor will be placed at the position of the error and the error message will be displayed. At this point the error may be fixed by editing the buffer. To return the cursor to the position of the current error after moving around the buffer the command `m2-err-current-error` can be issued.

To advance to the next error the command `m2-err-next-error` should be invoked again. The cursor will be moved to the new error position and the error message display. If no more errors exist the message "End Of Error List! (returning to first error)" will be displayed on the echo line. Selecting the `m2-err-next-error` command will again take the cursor to the first error in the source text.

The procedure for fixing errors after compiling a file using the `m2-comp-file` command is slightly different. Before EMACS can step through the errors found in the file, the file must read into a buffer using the command `find-file`. The name used to load the file must be identical to the name used to compile the file. An easy way to ensure the exact name will be used is to use the "exact-fname-mode". After the file is read into a buffer, the process used

to step through errors is the same as for source text compiled from a buffer.

By default EMACS will only step through the first 20 errors found in the source text, even if the compiler detected more than 20 errors. This number of errors recognized by EMACS can be changed using the command `set-m2-err-count`. The command prompts for the number of errors to handle. To have EMACS step through all of the errors specify an error count of zero.

When the compiler is loaded into EMACS it will occupy about 100K bytes of memory. When the compiler is invoked to perform a compilation additional memory for the compiler's heap, code buffer and other workspace is allocated. If there is not enough memory to allocate EMACS will display a message indicating that the compiler needs more memory. The command `free-mem-avail` is useful for checking the amount of memory currently available. Memory can be freed up by killing unused buffers. In some rare situations the memory in the Amiga may have been badly fragmented and while the `free-mem-avail` command may report enough memory free, the compiler may still not be able to compile. The only remedy to this problem is to exit EMACS and then return to EMACS, this should solve the problem, as a last resort rebooting the Amiga will always solve the problem.

The Modula-2 Linker

After a program has been compiled it may be linked to generate an executable file. The linking of a program can be performed using the linker called "M2Lk" from the CLI or the linker built into EMACS can be used for linking. The linker built into EMACS has exactly the same features as the stand alone linker program but provides facilities for linking by pressing just one key. For more information refer to the section of this manual which describes the linker.

Before the EMACS linker is used it may be necessary to initialize a few of the linker options. If most of the time the same program is being linked it may be useful to specify the name of the programs main module, so that linking can be done by just pressing one key. To set the name of the main module the command `set-m2-main-module` is used. The command prompts for the name of the main module. It may also be necessary to specify the names of the directories the linker should search when looking for files. The command `set-m2-obj-dir` prompts for a list of directory names each separated by a blank space. By default the linker outputs the executable file to the current directory, it may be directed to place executable files into another directory using the `set-m2-link-out-dir` command.

Assuming a main module has been specified, all that is needed to link a program is to invoke the `m2-link-main` command, this will start the linking process. The linker uses the echo for displaying information during the linking process. The name of each object module imported by the program being linked will be displayed with a preceding "<-" to indicate input. The linker will then display the symbol "->" followed by the name of the output file. After the linking is complete the linker will display the statistics of the generated program, refer to the linker section of this manual for an explanation of this

information.

The linker can also be directed to link a module of a specified name using the `m2-link-module` command. This command will prompt for the name of a module to be linked. After the name of the module is entered the linking process is identical to the one described above for the `m2-link-main` command.

Finally, if the executable file will be debugged using a symbolic debugger, the linker can be directed to include symbols in the executable file. The command `set-m2-link-gen-sym` is used to enable or disable adding symbols to the executable file or to add source level debugger information to the executable file. This command corresponds with the `"-d"`, `"-a"` and `"-g"` options of the stand alone linker. When symbol generation is enable, after the name of the execute file is displayed during the linking process the names of the symbol and reference files being read will be displayed on the echo line preceded with the name `"Sym:"`.

Running Modula-2 Programs

After a program has been successfully compiled and linked the program can be executed from inside EMACS, from the CLI or from the WorkBench. This section deals with running Modula-2 programs from EMACS.

Programs invoked from EMACS run concurrently with the editor and so editing may continue while a program is running. In addition more than one program may be invoked at the same time, the only limitation is the available memory. To prevent any possible problems, EMACS can not be exited while any user programs are still running.

To run the `"main"` module use the function `"m2-run-main"`, to run a module of a specified name use the command `"m2-run-module"`. Both functions look in the current directory or the linker output directory for an executable file which has the corresponding name. A window to be used for input and output is opened for the user program, the user program then begins execution. After the program terminates press `<RETURN>` to close the output window.

Other commands relating to running user programs are available, refer to the command summary at the end of this chapter for additional documentation.

Command Descriptions by Topic

In this section all of the commands available in EMACS are listed along with their descriptions. The commands are broken down by topic for quick reference. The default key binding for the command is listed in brackets after the name of the command, some rarely used commands are not bound to any key by default. The term "ARG" used in this section refers to the preceding command argument specified using Ctrl-U before the command.

Cursor Commands

backward-char [Ctrl-B]

Moves the point to the left (back) one character position.

backward-paragraph [Esc-[]

Moves the point backward to the beginning of the current paragraph.

backward-word [Esc-B]

Moves the point backward to the beginning of the current word.

beginning-of-buffer [Esc-<]

Set the mark at the current point position. Position the point on the first character in the buffer.

beginning-of-line [Ctrl-A]

Moves the point to the beginning of the current line.

end-of-buffer [Esc->]

Set the mark at the current point position. Position the point on the last character in the buffer.

end-of-line [Ctrl-E]

Moves the point to the end of the current line.

forward-char [Ctrl-F]

Moves the point one character position to the right (forward).

forward-paragraph [Esc-]]

Moves the point forward to the end of the current paragraph.

forward-word [Esc-F]

Moves the point forward to the end of the current word.

goto-line [Esc-G]

The command prompts for a line number. Moves the point to the beginning of the specified line. If the specified line number is greater than the number of lines in the buffer then moves the point to the last line of the buffer. The ARG may be used to specify the line number.

next-line [Ctrl-N]

Moves the point forward to the same column in the next line.

previous-line [Ctrl-P]

Moves the point backward to the same column in the preceding line.

Window Commands**delete-other-windows [Ctrl-X 1]**

Returns the editor display to one window by expanding the current window to fill the entire display.

delete-window [Ctrl-X 0]

Remove the current window from the display. Moves the point to the next window in the display.

enlarge-window [Ctrl-X ^]

Enlarge the size of the current window. The window above or below the current window will shrink to make room for the larger current window. The ARG may be used to specify the number of lines to enlarge the window.

next-window [Ctrl-X N or Ctrl-X 0]

Moves the point to the next(down) window in the display. If the current window is at the bottom of the display then moves to the window at the top of the display.

previous-window [Ctrl-X P]

Moves the point to the previous(up) window in the display. If the current window is at the top of the display then moves to the window at the bottom of the display.

recenter [Ctrl-L]

Recenter the window to make the current line the center line. The ARG value specifies the new position of the line relative to the top of the window.

scroll-down [Esc-V]

Moves the window backward in the edit buffer by about one window length. The window is positioned on the edit buffer so that the previous first line in the window becomes the last line. The ARG may be used to specify the number of lines to scroll.

scroll-other-window [Esc Ctrl-V]

Moves the next(down) window in the display forward in the edit buffer by about one window length. The window is positioned on the edit buffer so that the previous last line in the window becomes the new first line. The ARG may be used to specify the number of lines to scroll.

scroll-up [Ctrl-V]

Moves the window forward in the edit buffer by about one window length. The window is positioned on the edit buffer so that the previous last line in the window becomes the first line. The ARG may be used to specify the number of lines to scroll.

shrink-window

Shorten the size of the current window. The window above or below the current window will become larger to fill in the gap in the display. The ARG may be used to specify the number of lines to shrink the window.

split-window-vertically [Ctrl-X 2]

Splits the current window into two windows, initially both windows will display the same buffer. This operation will not work if the current window is smaller than three lines in height.

Buffer Commands

insert-buffer

The command prompts for the name of the buffer. The contents of the specified buffer is inserted at the current point position. Pressing the <RETURN> key without entering a name will select the alternate buffer if one exists.

kill-buffer [Ctrl-X K]

The command prompts for the name of the buffer. Enter the name of the buffer to kill or press <RETURN> to kill the current buffer. The contents of the specified buffer is discarded and the buffer itself is deleted from the system. When killing the current buffer the contents of the window will change to display the alternate buffer. If the buffer has been modified but not saved EMACS will request a confirmation.

list-buffers [Ctrl-X Ctrl-B]

Creates a new buffer called "*Buffer List*", the contents of the new buffer is a list. Each line in the buffer list represents a buffer and the status information for that buffer. If the first character of the line contains "." then this is the current buffer. The first column in the list labelled "MR" may contain a "*" symbol to indicate that the contents of the buffer has been changed, but the change has not been written out to a file. The second column labelled "Buffer" contains the name of a buffer. The third column labelled "Size" contains a value specifying the number of character in the buffer. The last column labelled "File" contains the file path specification for the file associated with the buffer. The new buffer containing the list of buffers is displayed in one of the existing windows or a new window is created, if only one window exists on the display.

not-modified [Esc-~]

Marks the current buffer as unmodified since it was last read or written to a file. Clears the buffer status "***" in the mode line.

switch-to-buffer [Ctrl-X B]

Selects a specified buffer and displays it in the current window. The command prompts for the name of the desired buffer. Pressing the <RETURN> key without entering a buffer name selects the alternate buffer. If the specified buffer name does not exist a new buffer is created and the new name is assigned to the buffer.

switch-to-buffer-other-window [Ctrl-X 4 B]

Selects a specified buffer and displays it in another window. The command prompts for the name of the desired buffer. Pressing the <RETURN> key without entering a buffer name selects the alternate buffer. If the specified buffer name does not exist a new buffer is created and the new name is assigned to the buffer.

File Commands**current-dir**

The command displays the default directory for EMACS. The default directory may be changed by specifying a new path. Pressing <RETURN> exits without changing the current directory.

exact-fname-mode

Toggles exact filename mode on or off. When the exact filename mode is enabled the exact case of the filename on disk is used rather than the user specified case. This mode is initially disabled.

file-req-mode

Toggles between file request window mode and keyboard input mode for filename input. Initially the file request window mode is active. When active it causes a file request window to be opened whenever EMACS requires a filename to be specified. When inactive it causes a prompt to appear on the echo line requiring the user to type in the filename.

find-file [Ctrl-X Ctrl-F or F9]

The command prompts for the name of a file to edit or creates a file request window, determined by the state of the "file-req-mode". The list of buffer names is searched for the specified filename. Selects the buffer with that filename if there is one. Otherwise, creates a buffer with that name and reads the file into the new buffer from disk.

find-file-other-window [Ctrl-X 4 F or Shift-F9]

The command prompts for the name of a file to edit or creates a file request window, determined by the state of the "file-req-mode". The list of buffer names is searched for the specified filename. Selects the buffer with that filename if there is one. Otherwise, creates a buffer with that name and reads the file into the new buffer from disk. The buffer will be displayed in another existing window or a new window will be created for the buffer.

insert-file [Ctrl-X I]

The command prompts for the name of a file to insert or creates a file request window, determined by the state of the "file-req-mode". The contents of the specified file is inserted after the point in the current buffer. The mark is set to the character after the inserted text.

make-backup-files

Toggle file backup mode on or off. When the file backup mode is enabled

and the find-file command is invoked the buffer will be marked for backup. If the file backup mode is enabled when the file is saved then the file on disk will be renamed to a new filename with the character '~' appended to the name. On subsequent save file operations on that buffer the backup file will not be changed. Using the file backup mode allows the previous version of a file to be recovered. To disable the file backup mode invoke the command with a preceding ARG. This mode is initially disabled.

save-buffer [Ctrl-X Ctrl-S or F10]

Copies the contents of the current buffer into the file associated with the current buffer. The buffer is not written out if no changes have been made since the last save operation or since the file was read into the buffer. The buffer must have an associated filepath in order for this command to work.

save-buffers-kill-emacs [Ctrl-X Ctrl-C]

Exits the editor returning to the dos shell. A prompt will be issued for each buffer which has been modified but not written out to a file, the prompt asks if the contents of the buffer should be written out to a file. If after presenting a prompt for each modified buffer any buffers modified buffers still remain a final prompt will appear requesting a confirmation for the exit from the editor. Only buffers with filepaths will be checked for modifications.

save-some-buffers [Ctrl-X S]

A prompt will be issued for each buffer which has been modified but not written out to a file, the prompt asks if the contents of the buffer should be written out to a file. The command may be used with an ARG in which case all modified buffers will be written out without issuing any request prompts. Only buffers with filepaths will be checked for modifications.

write-file [Ctrl-X Ctrl-W or Shift-F10]

The command prompts for a filename to be used for writing out a file or creates a file request window, determined by the state of the "file-req-mode". The contents of the current buffer is written out to the specified file.

Copy/Delete/Kill Commands

backward-delete-char [Ctrl-H or BackSpace]

Delete the character to the left of the point. The command may be used with ARG to delete a specified number of characters and place into kill buffer.

backward-kill-word [Esc Backspace]

Moves the word to the left of the point to the kill buffer.

copy-region-as-kill [Esc-W]

The characters between the current point and mark are copied to the kill buffer.

delete-blank-lines [Ctrl-X Ctrl-0]

If the point is on a blank line then any blank lines above or below the current line including the current line are deleted from the buffer. If the point is on a non blank line then any blank lines below the current line will be deleted.

delete-char [Ctrl-D or DEL]

Deletes the character to the right of the point. The ARG may be used to kill a specified number of characters and place the characters into the kill buffer.

kill-line [Ctrl-K]

Moves all characters to the right of the point on the current line to the kill buffer, not including the end of line character. If the end of line character is the only character to the right of the point on the current line, it is moved to the kill buffer.

kill-paragraph

The current paragraph is deleted and placed into the kill buffer.

kill-region [Ctrl-W]

The characters between the current point and mark are delete from the buffer and copied into the kill buffer.

kill-word [Esc-D or Esc DEL]

Moves the characters from the point until the end of the current word, to the kill buffer.

yank [Ctrl-Y]

Insert the contents of the kill buffer at the point. After inserting the text the point is moved to the next character after the inserted text.

Search/Replace Commands

isearch-backward [Ctrl-R]

Search backwards from the point incrementally. The Incremental search causes the current buffer to be searched for a string as it is being entered, unlike the non-incremental search which waits for the entire search string to be specified. At the prompt typing a normal character will add that character to the search string, or one of the following commands may be entered:

- * BackSpace - Undo the last command typed.
- * Esc - Exits incremental search, leaving the point at the current location.
- * Ctrl-S - Search again forward from the current location.
- * Ctrl-R - Search again backwards from the current location.
- * Ctrl-Q - Add a quoted character to the search string.
- * Ctrl-G - If the current search failed, will return the search string to the last successfully found string. If the current search is successful will cancel the command and return point to the original position.
- * Other control characters terminate the incremental search and then are executed normally.

isearch-forward [Ctrl-S]

Search forward from the point incrementally. The Incremental search causes the current buffer to be searched for a string as it is being entered, unlike the non-incremental search which waits for the entire search string to be specified. At the prompt typing a normal character will add that character to the search string, or one of the following commands may be entered:

- * BackSpace - Undo the last command typed.
- * Esc - Exits incremental search, leaving the point at the current location.
- * Ctrl-S - Search again forward from the current location.
- * Ctrl-R - Search again backwards from the current location.
- * Ctrl-Q - Add a quoted character to the search string.
- * Ctrl-G - If the current search failed, will return the search string to the last successfully found string. If the current search is successful will cancel the command and return point to the original position.
- * Other control characters terminate the incremental search and then are executed normally.

query-replace [Esc-%]

The command prompts for the string to be replaced and the replacement string. The search for the string begins at the point and continues to the end of the current buffer. Each time the string to be replaced is found the following commands are available:

- * Help - Display the available options.
- * Space - Replace original string with the replacement string.
- * '.' - Replace original string with the replacement string and then exit the query-replace command.
- * BackSpace - Advance to the next occurrence of the original string.
- * '!' - Replace all the remaining occurrences of the original string, without prompting.
- * Esc - Exit the query-replace command.

search-again [Esc-A]

Advance point to the next occurrence of the string which was used in the previous `isearch-forward`, `isearch-backward`, `search-forward` or `search-backward` command. The direction of the search for the next occurrence depends on the direction of the previous search command.

search-backward [Esc-R]

Prompts for a search string, if <RETURN> is pressed uses the search string from the previous search command. Searches for a specified string from the point to the beginning of the current buffer.

search-forward [Esc-S]

Prompts for a search string, if <RETURN> is pressed uses the search string from the previous search command. Searches for a specified string from the point to the end of the current buffer.

Case Commands**capitalize-word [Esc-C]**

If the first character of the next word is a letter, then convert the letter to upper case. The point is moved to the end of the word.

downcase-region [Ctrl-X Ctrl-L]

Convert the characters in the region from the point to the mark to lower case, if they are upper case letters.

downcase-word [Esc-L]

Convert the characters in the word to the right of the point to lower case. The point is moved to the end of the word.

upcase-region [Ctrl-X Ctrl-U]

Convert the characters in the region from the point to the mark to upper case, if they are lower case letters.

upcase-word [Esc-U]

Convert the characters in the word to the right of the point to upper case.

The point is moved to the end of the word.

Key Binding Commands

describe-bindings

Creates a new buffer called "*Help*", the contents of the new buffer is a list of the current key bindings. Each line in the buffer begins with the symbolic name of a key followed by name of the function currently associated with that key. The new buffer containing the key bindings is displayed in one of the existing windows or a new window is created, if only one window exists on the display.

describe-key-briefly [HELP]

The command prompts for a key and displays the function associated with that key.

global-set-key

The command prompts for a key and a function. The specified function becomes bound to the key.

global-unset-key

The command prompts for a key. The specified key is disassociated from the function previously attached to the key.

help

This command provides two kinds of help: pressing "B" executes the describe-bindings command or pressing "C" executes the describe-key-briefly command.

Macro Commands

call-last-kbd-macro [Ctrl-X E]

Execute the currently defined keyboard macro. If a macro has not been defined yet then nothing happens.

end-kbd-macro [Ctrl-X)]

Stop recording keyboard input for macro. The macro is now recorded and can be invoked using the call-last-kbd-macro function.

start-kbd-macro [Ctrl-X (]

Begin recording keyboard input for macro. Each key sequence entered is executed normally, but is also recorded for later execution as a macro.

Fill Mode Commands

auto-fill-mode

Toggle word-wrap at end of line mode, this mode effects all further input into any buffer. When text is being entered the <RETURN> key does not have to be entered, words will automatically be wrapped to the next line after

reaching the display column designated as the fill column. This mode may be useful for using the editor as a limited word processor. The <RETURN> key will only be used at the end of paragraphs. The mode line will display the word "fill" while the editor is in this mode.

fill-paragraph [Esc-Q]

Format the lines above and below the current line which contain words that begin on the first character of the line, stop when a line is reached which does not have a character in the first position. The first line of the paragraph may begin with blank spaces. If this command is issued on a blank line then move forward in the buffer until a line containing words is reached. The words contained in the lines are formatted into a paragraph between the left edge and the fill column. The point is moved to the end of the new paragraph.

insert-with-wrap

This function is bound to the <Space> key when in auto-fill-mode. The function checks the current column and adds a newline character if past the fill column.

set-fill-column [Ctrl-X F]

Specify the display column to be used as the fill column, this column will serve as the left margin of paragraphs. The current cursor position is used as the fill column designator. The fill column may also be specified by ARG.

Expression Commands

eval-current-buffer

The contents of the current buffer is assumed to contain expressions to be evaluated. The expressions in the buffer are evaluated, each expression is an EMACS function.

eval-expression

The command prompts for an expression. The expression is evaluated and executed, each expression is an EMACS function.

load

The command prompts for the name of the file to evaluate. Load and evaluate a file containing expressions, each expression is an EMACS function. The contents of the file is not loaded into a buffer.

Miscellaneous Commands

auto-indent-mode

Toggle auto indent mode, this mode causes the point to be placed on the same column as the first word in the first non blank line above the current line. This operation occurs after pressing the <RETURN> key at the end of a line. This function works by swapping the bindings of the <RETURN> key and Ctrl-j key.

blink-matching-paren

Causes EMACS to check for balanced parenthesis. This means each "(" must have a corresponding ")". The check is performed at the time the character ")" is pressed. If the corresponding "(" is visible in the current window the cursor is moved to the "(" character and then returned to the original position. If the corresponding "(" is not visible in the current window the line which contains the matching "(" is displayed on the echo line. Finally, if the user types ")" and a matching "(" character can not be found, the display will blink to alert the user. To disable the checking for balanced parenthesis, invoke this command with a preceding ARG set to zero.

blink-matching-paren-hack

When the blink-matching-paren is invoked the character "(" is rebound to this command rather than to the self-insert-command. EMACS can also be forced to check for balance of the following characters "[]", "{ }" or "< >". To cause EMACS to check for one of these other characters bind the right character of the pair to this command.

ctrlx-four-hack [Ctrl-X 4]

This command provides support for commands which begin with Ctrl-X 4. This command is normally executed in a sequence with other keys to direct an action to occur in another window.

delete-horizontal-space [Ctrl-\]

Deletes the white space characters(spaces and tabs) to the left and right of the point until a non white space character is encountered or until the end or the beginning of the line is reached.

exchange-point-and-mark [Ctrl-X Ctrl-X]

Exchange the point with the mark. The command will fail if no mark has been set in the current window. This command is useful for jumping to the beginning and to the end of a region.

execute-extended-command [Esc-X]

The command prompts for the name of a function to execute. The purpose of this function is to allow the execution of functions not currently bound to any keys or if the user cannot remember the key associated with a given function.

free-mem-avail [Shift-F5]

Display on the echo line the amount of free memory currently available in the system. Due to memory fragmentation there may not be enough contiguous memory to perform the desired operation even when the free memory display indicates sufficient memory is available.

insert-newline [Return]

This command inserts a newline character, it is normally bound to the return key.

just-one-space [Esc-Space]

Deletes all white space character(spaces and tabs) to the left and right of the point until a non white space character is encountered or until the end or the beginning of the line is reached. Then insert one white space character at the point.

keyboard-quit [Ctrl-G]

This command may be used from almost anywhere in EMACS to abort the current operation. This command is not valid in the file requester window.

newline-and-indent [Ctrl-J]

Insert a newline character and advance the point to the next line, the point is placed on the same column as the first non space character on the first non-blank line above the current line. Performs the same function as the <RETURN> key while in auto-indent-mode.

open-line [Ctrl-O]

Insert a newline character at the current point position. This effectively splits the line at the position of the point.

overwrite-mode

Toggle overwrite mode, this mode causes characters typed into a buffer to overwrite the characters already in the buffer rather than inserting new characters. However, when at the end of a line or the end of a buffer, characters typed are inserted as if EMACS was in insert mode.

prefix-region

Prefix each line between the point and the mark with the string specified by the set-prefix-string command. If an ARG is used the command will prompt for the string to be used instead of using the currently specified prefix string. The default prefix string is ">".

redraw-display

Forces EMACS to redraw the entire display. This command may need to be executed to restore the display after it has become damaged.

quoted-insert [Ctrl-Q]

The next key pressed is taken literally and inserted into the buffer. This command allows the entry of any control character key combination.

self-insert-command

Insert the character associated with the given key at the point. This function is normally bound to all of the normal typewriter keys.

set-mark-command [Ctrl-@]

Set the mark at the current position of the point.

set-prefix-string

Specify the string to be used by the prefix-region command. The specified string may be later used by the prefix-region command to prefix lines with a given string. The default prefix string is ">".

suspend-emacs [Ctrl-Z]

Open a new AmigaDOS command line interface (CLI) in a new window. Since the AmigaDOS operating system supports multi-tasking, the editor can continue to be used after issuing this command.

transpose-chars [Ctrl-T]

The character at the current point position is swapped with the previous character. If this command is issued at the beginning of a line, nothing happens.

what-cursor-position [Ctrl-X =]

Display information on the echo line about the position of the cursor. The following is an example of the information which may be displayed and the meaning of each component:

Char: A (0101) point=7(17%) line=2 row=2 col=5

A	- character at the current position
(0101)	- an octal code for the character
point	- character position relative to start of buffer
(17%)	- character position as percentage of buffer size
line	- the line relative to the beginning of the buffer
row	- the row relative to the top of the window
col	- the column relative to the top of the window

Mouse Commands

amiga-mouse

Set point to the character that is closest to where the mouse pointer is located. If the point is already at the location of the mouse then set the mark to the position of the point.

**mouse-recenter, mouse-kill-word, mouse-kill-line,
mouse-delete-char, mouse-just-one-space, mouse-kill-region,
mouse-yank**

These commands are executed by a mouse click in the text area of a window. The point is moved to the current position of the mouse pointer and then the command of the specified name is executed. For example "mouse-recenter" would execute the "recenter" command.

**mouse-scroll-up, mouse-scroll-down, mouse-split-window-vertically
mouse-delete-window, mouse-beginning-of-buffer,
mouse-end-of-buffer, mouse-enlarge-window, mouse-shrink-window**

These commands are executed by a mouse click on the mode line of a window. The point is restored to the its original position for the window with which the mode line is associated and then the command of the specified name is executed. For example "mouse-scroll-up" executes the command "scroll-up".

Display Setup Commands

set-mode-background

The command prompts for the background color to be used for displaying the mode line. The background color may be specified as a value between zero and seven.

set-mode-foreground

The command prompts for the foreground color to be used for displaying the mode line. The foreground color may be specified as a value between zero and seven.

set-mode-rendition

The command prompts for the attributes to be used for displaying the mode line. The available attributes are:

- 0 - plain
- 1 - bold
- 3 - italic
- 4 - underline
- 7 - reverse video

set-text-background

The command prompts for the background color to be used for displaying text. The background color may be specified as a value between zero and seven.

set-text-foreground

The command prompts for the foreground color to be used for displaying text. The foreground color may be specified as a value between zero and seven.

set-text-rendition

The command prompts for the attributes to be used for displaying text. The available attributes are:

- 0 - plain
- 1 - bold
- 3 - italic
- 4 - underline
- 7 - reverse video

toggle-window-hack

Toggles EMACS window between a non-resizable window(borderless) and a resizable window. When the window is borderless the window will occupy the entire display. When toggled back to a window with borders it will return to its previous size.

Modula-2 Compiler Commands

m2-comp-load [F8]

The command prompts for the argument string to be passed to the compiler. To load the compiler the command searches the current directory for the file "M2". If the compiler is not found in the current directory the AmigaDOS PATH will be searched and finally the "C:" directory is searched. After the compiler is loaded a message "Compiler Loaded." is displayed or an appropriate error message is displayed if a problem is encountered.

m2-comp-unload [Shift-F8]

Removes the compiler from memory if it is currently loaded. The memory occupied by the compiler may now be freely used for other purposes. This command is automatically executed when exiting the editor.

m2-comp-current-buffer [F2]

The current buffer is compiled. After compilation a message is displayed indicating the number of errors encountered. If the compilation is successful a symbol or object file is generated.

m2-comp-buffer [Shift-F2]

The command prompts for the name of the buffer to be compiled. The specified buffer is compiled. After compilation a message is displayed indicating the number of errors encountered. If the compilation is successful a symbol or object file is generated.

m2-comp-file [F5]

The command prompts for the name of the file to be compiled or creates a file request window, determined by the state of the "file-req-mode". The specified file is compiled. After compilation a message is displayed indicating the number of errors encountered. If the compilation is successful a symbol or object file is generated.

m2-err-current-error [F1-Shift]

Moves the point to the location of the current compilation error. The error message associated with the error is displayed on the echo line and placed into the title bar of the window. The buffer containing the error is displayed in the current window, if it is not already there.

m2-err-next-error [F1]

Moves the point to the location of the next compilation error. The error message associated with the error is displayed on the echo line and placed into the title bar of the window. The buffer containing the error is displayed in the current window, if it is not already there.

set-m2-comp-opt

This command is used to control various compiler options interactively. The changes made by this command are temporary and only stay in effect while the current instance of the compiler is loaded. If EMACS is exited or the compiler is unloaded the changes are discarded. The default value for each compiler options is displayed, a new value may be specified or press <RETURN> to skip to the next option. If an invalid value is

specified the display will flash.

set-m2-comp-stack-size

The command prompts for the amount of stack space to be allocated for the compiler. Pressing <RETURN> without entering a new number, will keep the same size as previously set. This command may only be issued while the compiler is not loaded. By default the stack size is set to 10000.

set-m2-err-count

The command prompts for a value specifying the maximum number of errors to be remembered after a compilation. By default this number is 20, meaning that only the first 20 compilation errors will be remembered. To remember all the errors generated by the compiler set the error count to zero.

Modula-2 Linker Commands

m2-link-main [F3]

Perform a link operation on the module currently specified as the "main" module. If the "main" module has not been specified with the set-m2-main-module command an error message will be displayed. The specified module is linked with the necessary library modules and an executable file is generated.

m2-link-module [Shift-F3]

The command prompts for the name of the module to be linked. The specified module is linked with the necessary library modules and an executable file is generated.

set-m2-link-out-dir [Shift-F6]

The command prompts for the name of the directory to which the executable file generated by the linker should be sent. To reset the output directory back to the current directory press <RETURN>.

set-m2-link-gen-sym [Shift-F7]

This command is used to control certain linker options. Each option is specified as a TRUE or FALSE value. At each prompt type "T" to set the option to TRUE, "F" to set the option to FALSE, <RETURN> to advance to the next prompt without changing or Ctrl-g to abort without making a change. Three prompts are presented for examining or changing options. The first one asks if special information for use with a source level debugger should be generated. The second prompt asks if symbols should be added to the executable file for symbolic debugging, and the last prompt asks if symbols added to the executable should be prefixed with the name of the module. If the source level debug information option is set to TRUE then the second option to generate symbols should be set to FALSE and vice versa.

set-m2-main-module [F6]

The command prompts for the name of the "main" module, the module which is the root of the Modula-2 program being worked on. The m2-link-main command may be used to link the "main" module without having to specify the name of the module each time a link operation is needed.

set-m2-obj-dir [F7]

The command prompts for one or more names of directories which are to be searched for library modules during the linking process. The directories are listed on the same line, separated by a space character.

Modula-2 Run Commands

m2-run-main [F4]

Run the module currently specified as the "main" module. An output window is created and the executable file with the same name as the "main" module is invoked. The executable file is loaded from the linker's output directory. After the program finishes type <RETURN> in the output window to close the window. The user program is executed in parallel with EMACS so editing may continue even while the program is running. Any number of user programs may be executed simultaneously.

m2-run-module [Shift-F4]

This command prompts for the name of a module to be executed. An output window is created and the executable file with the same name as the specified module is invoked. The executable file is loaded from the linker's output directory. After the program finishes type <RETURN> in the output window to close the window. The user program is executed in parallel with EMACS so editing may continue while the program is running. Any number of user programs may be executed simultaneously.

set-m2-run-args

This command prompts for an argument string to be passed to user programs invoked from EMACS. This arg string is passed to the user program as if the program was executed from the CLI with arguments.

set-m2-run-stack-size

This command displays the default stack size specified for user programs. A new stack size value may be specified or press <RETURN> to keep the current default. This command should be used with care, if the stack size value is too small for the program being run the system will probably crash. By default the stack size is set to 20000 bytes.

set-m2-run-window

This command display the default output window specification. A new window specification may be specified or press <RETURN> to keep the current default value. A window specification may specify a "RAW:" or "CON:" window, for more information on this refer to the AmigaDOS technical documentation. The window specification must end with a "/" character. The command does not check the validity of the window specification. However, an error message will be displayed when a program is actually invoked with an invalid window specification. The following are examples of valid window specifications: "RAW:0/0/640/200/" or "CON:0/0/100/100/".

Chapter 7	Modula-2 Error Lister.....	3
	Running the Error Lister.....	3

Chapter 7 Modula-2 Error List	3
Running the Error List	3

This chapter describes the error lister program included in this package. The purpose of the error lister is to show the locations in the source file where the compiler has detected errors. It should be noted that most users will be using the integrated EMACS environment with its built in error tracking capability. The stand alone error lister is provided as an alternative for users who may wish to use an editor other than the EMACS editor included with the package. The other possible use for the stand alone error lister, is when performing batch compilation jobs, to examine the errors from all of the compiles.

For additional information on the error messages refer to the section which describes the error messages which the compiler generates.

Running the Error Lister

The error lister program is called "M2Err". The syntax for the error lister is as follows:

```
M2Err [Options]
```

When the error lister is invoked it reads an error log file called "M2.err" from the directory "T:". If the compiler has not been used to compile a source file, the error log file will not exist. The error lister will display a message if it can not find the error log file.

If an error log file exists the error lister will list the errors encountered for each source file compiled. The listing of errors is in one of two formats. The format used is dependent on the ability of the error lister to find the source file associated with the errors.

If the error lister finds the source file then each error is displayed in the following format:

```
Flag := TestCond;          <-- source line
                          <-- error position in line
line: 10 err: 050 identifier not declared on not visible <-- info
```

If the error lister can not find the source file then each error is displayed in the following format:

```
pos: 000021 err: 050 identifier not declared or not visible
```

The "pos" field is the byte offset of the error relative to the beginning of the file.

Modula-2 Software Construction Set Reference Guide

The error lister allows several command line options. Each option begins with the character "-" followed by a letter. The following is a list of the available options:

* -w

Open a window to which the error list will be output. Choosing this option implicitly chooses the "-p" option to pause after each error is displayed.

* -p

Pause after each error is displayed. To continue press the <RETURN> key.

Running the Error Lister

The error lister program is called "M2ERR". The syntax for the error lister is as follows:

M2ERR [options]

When the error lister is invoked it reads an error log file called "M2.ERR" from the directory "F:". If the compiler has not been used to compile a source file, the error log file will not exist. The error lister will display a message if it can not find the error log file.

If an error log file exists the error lister will list the errors encountered for each source file compiled. The listing of errors is in one of two formats. The format used is dependent on the ability of the error lister to find the source file associated with the errors.

If the error lister finds the source file then each error is displayed in the following format:

```
File: 10 err: 000 identifier not declared on not visible <--- info
Flag: 10 testCond:
      <--- error position in line
      <--- source line
```

If the error lister can not find the source file then each error is displayed in the following format:

```
pos: 000001 err: 000 identifier not declared on not visible
```

The "pos" field is the byte offset of the error relative to the beginning of the file.

Chapter 8	Modula-2 Compiler Configuration Utility.....	3
	Running the Configuration Utility.....	3

Chapter 8: Modula-2 Compiler Configuration Utility..... 8
Running the Configuration Utility..... 8

This chapter describes a utility which may be used for customizing the configuration of the Modula-2 compiler. The configuration utility modifies the compiler executable file with the specified values. If you wish to change a compiler option temporarily it may be more practical to specify a compiler command line option. However, to change the compiler configuration permanently use the configuration utility.

Running the Configuration Utility

The name of the configuration utility is "M2Config". The syntax for running the configuration utility is as follows:

```
M2Config [Options] <M2_FileName>
```

The <M2_FileName> parameter specifies the filename of the Modula-2 compiler to be configured. If the compiler is not in the current directory then the full path to the compiler must be specified.

The configuration utility allows several command line options. Each options begins with the character "-" followed by a letter. The following is a list of the available options:

* -d

Reset the compiler configuration to the default configuration values. The default configuration may use values which are too large for an Amiga with only 512K of memory.

* -s

Display the current compiler configuration without prompting the user to change the current configuration values.

Examples:

```
M2Config c:M2           ; Change configuration
M2Config -s c:M2        ; Show configuration
M2Config -d c:M2        ; Reset to default configuration
```

If no options are specified to the M2Config utility then the current configuration is displayed and a prompt is given for each compiler parameter.

Modula-2 Software Construction Set Reference Guide

The current compiler configuration is displayed as follows, the actual values may vary from the ones listed below:

```
Compiler FileName: c:M2
Heap Size.....150000
Code Buffer Size.....32766
Constant Buffer Size.....32766
Identifier Buffer Size.....32766
Disk Buffer Size.....1024
Address Relocation Buffer Size.....1000
Range Checking.....FALSE
Overflow Checking.....FALSE
Global Variables Absolute Addressing...FALSE
Generate Reference File.....FALSE
Clear Module Key.....FALSE
```

The values displayed represent the current configuration of the compiler. You will now be prompted to enter a new value for each of the above configuration parameters. The value listed in brackets is the current value. To avoid changing a parameter press <RETURN> when prompted to enter a new value.

The numerical parameters in the compiler configuration may need to be made smaller if the compiler is going to be used from within the EMACS editor on an Amiga with only 512K of memory.

A suggested compiler configuration for a 512K Amiga is as follows:

```
Heap Size                80000
Code Buffer Size          10000
Constant Buffer Size      1000
Identifier Buffer Size    17000
Disk Buffer Size          1024
Address Relocation Buffer Size 200
```

These values are only suggested and may need to be adjusted based on the size of the Modula-2 programs actually compiled.

The amount of memory actually required by the compiler can be calculated by summing the above specified compiler configuration values and adding in the size of the compiler program and its stack. It should be noted that the memory specified by the compiler configuration is only used while actually compiling so that while working with the EMACS editor performing editing the amount of memory actually used by the compiler is approximately 100K.

```
Compiler Program          92000
Compiler Stack            10000
Sum of the Configuration Values  xxxx
-----
Total Amount of Memory Used  xxxx
```

Chapter 9 Quick Load Utility.....	3
Creating a File List.....	3
Loading a Quick Load File.....	4
Making a Quick Load File.....	4

Chapter 2 Quick Load Utility.....	2
Creating a File List.....	3
Loading a Quick Load File.....	4
Making a Quick Load File.....	4

This chapter describes the quick-load utility included with this package. The purpose of the quick-load program is to provide a very fast way of moving files to a ram disk device. The technique allows a very large number of files to be moved to a ram disk in a fraction of the time that would be required using the AmigaDOS "copy" command.

The name of the quick-load program is "QLoad". The QLoad utility reads a specially formatted archive file called a quick-load file. The contents of the quick-load file is extracted and placed into a user specified directory usually residing on the ram disk. A quick-load file is created once using the same QLoad utility. Thereafter the files contained in the quick-load file may be loaded into ram at an amazingly fast speed. The primary use of this utility is to copy the Modula-2 libraries to the ram disk, the quick-load file containing all of the Modula-2 libraries may be found on the library disk and is called "M2Lib". The other advantage of this utility is when a large number of files are stored in the quick-load file format they require much less disk space than if each file was stored separately. To save memory it may be useful to build a smaller version of the "M2Lib" file containing only the Modula-2 libraries which you actually use.

Creating a File List

In order to make a quick-load file a text file containing the names of all the files to be included must be created. An easy way to accomplish this is to use the AmigaDOS "List" command with the "Quick" option which lists the names of the files in a given directory without any other information. Direct the output of the "List" command to an output file using output redirection. The resulting output file should then be loaded into EMACS. The "List" command sometimes adds a header line and always adds a line to the end of the list, these extra lines should be eliminated from the text file and the file should be saved back to disk. It should be noted that the QLoad program does not handle file names with imbedded spaces!

The following is an example of how a file list can be created using the AmigaDOS "List" command. Be sure to edit the file as described above before using it as a file list for the QLoad utility.

```
List >ram:LibList df1:M2L/ quick
```

The operation of the QLoad utility may be interrupted by the user by pressing the Ctrl-C key combination. The QLoad utility will return to the CLI without completing the operation being performed.

The QLoad utility has two different command line formats. The command line option "-m" is used to switch between the formats, this option must be the first parameter on the command line.

Loading a Quick Load File

QLoad <QuickFile> <OutputPath>

The first format is used to load a quick-load file. The first parameter <QuickFile> specifies the filename of a quick-load file to be loaded. The second parameter <OutputPath> specifies the destination directory for the load operation, this parameter must end on either the ":" character or the "/" character.

Examples:

QLoad df0:M2Lib ram:M2L/

QLoad df0:Cmds ram:

Making a Quick Load File

QLoad -m <FileList> <QuickFile>

The second format is used to make a quick-load file. The first parameter is the option "-m" designating the make mode. The second parameter specifies the name of the text file which contains the names of the files to be placed into the quick-load file. The third parameter specifies the name of the quick-load file to be created.

When making a quick-load file it is recommended that the output be sent to ram disk and if possible the files to be included should be on ram disk. If the make operation is done entirely on floppy and the number of files is large the operation may take a long time.

Examples:

QLoad -m ram:LibList ram:NewLib

QLoad -m ram:CmdList ram:Cmds

Chapter 10	Modula-2 Procedure Statistics Utility.....	3
	Running the Statistics Utility.....	3

Chapter 10 Modula-2 Procedure Statistics Utility.....3
Running the Statistics Utility.....3

This chapter describes a utility which may be used in determining which procedures in a Modula-2 program are being called most of the time. The output of this utility is a file containing a count of the number of times each procedure was called. The list of procedures is sorted from the most called to the least called. Procedures which are never called are dropped from the list automatically.

Running the Statistics Utility

The name of the utility program is "M2Stat". The command line syntax is as follows:

```
M2Stat <ProgFileName> [ProgParameters]
```

The <ProgFileName> specifies the name of the program to be analyzed. If the program is not in the current directory then the full path should be specified. The [ProgParameters] are any parameters which should be passed to the program being analyzed.

For the M2Stat utility to work properly the program being analyzed must be compiled and linked with symbols. For instructions on how to add symbols to a program refer to the appropriate sections of the compiler and linker chapters. The process of adding symbols is the same as would be used for debugging a program with a symbolic debugger.

When analyzing a program with the M2Stat utility it may run slower than it would normally run by itself. The degree to which the program will be effected depends on the specific program and to what extent it uses procedure calls.

After the program being analyzed exits the M2Stat utility will create a text file in the directory from which the program was executed. The text file contains a line for each procedure called along with the number of times that procedure was called. The name of the generated output file is taken from the name of the program being analyzed and adding the extension ".stat".

Examples:

```
M2Stat Draw 640 200 3  
M2Stat RayTrace
```

Chapter 11	Symbol File Cross Reference Utility.....	3
	Running the Cross Reference Utility.....	3

Chapter 11 Symbol File Cross Reference Utility.....3
Running the Cross Reference Utility.....3

This chapter describes the symbol files cross reference utility program. This cross reference utility unlike other cross reference utilities does not read a source text file extracting the identifiers, rather it reads compiler generated symbol files. The other unique feature of this program is that it can perform a batch operation processing many files at one time. The AmigaDOS "Sort" utility can be used to sort the output of the cross reference utility.

Running the Cross Reference Utility

The name of symbol cross reference program is "M2SymXRef". The command line syntax is as follows:

```
M2SymXRef <FileList> <OutputFile>
```

The first parameter <FileList> is a text file containing the names of all the files to be processed. An easy way to create a file list is to use the AmigaDOS "List" command with the "Quick" option which lists the names of the files in a given directory without any other information. Direct the output of the "List" command to an output file using output redirection. The resulting output file should then be loaded into EMACS. The "List" command sometimes adds a header line and always adds a line to the end of the list, these extra lines should be eliminated from the text file and the file should be saved back to disk. It should be noted that the M2SymXRef utility does not handle file names with imbedded spaces! The second parameter <OutputFile> designates the file to which output of the cross reference utility should be sent.

The following is an example of how a file list can be created using the AmigaDOS "List" command. Be sure to edit the file as described above before using it as a file list for the M2SymXRef utility.

```
List >ram:LibList df1:M2L/ quick
```

The output file created by M2SymXRef contains a line for each identifier, with the following information on each line.

```
<Identifier> <ModuleName> <ObjectClass>
```

```
<Identifier>      - Identifier
<ModuleName>     - Definition appears in this module
<ObjectClass>    - CONST,TYPE,VAR,PROCEDURE
```

The following is a short sample of the output from the M2SymXRef utility.

```
Lookup      FileSystem      PROCEDURE
Close       FileSystem      PROCEDURE
```

Examples:

```
M2SymXRef LibList sxref      ; Generate xref
Sort sxref TO sxref.sorted   ; Sort xref
```

This chapter describes the symbol file cross reference utility program. This cross reference utility unlike other cross reference utilities does not read a source text file extracting the identifiers, rather it reads compiler generated symbol files. The other unique feature of this program is that it can perform a batch operation processing many files at one time. The AmigaDOS "sort" utility can be used to sort the output of the cross reference utility.

Running the Cross Reference Utility

The name of symbol cross reference program is "MS2SYMXREF". The command line syntax is as follows:

```
MS2SYMXREF <FileList> <OutputFile>
```

The first parameter <FileList> is a text file containing the names of all the files to be processed. An easy way to create a file list is to use the AmigaDOS "list" command with the "quick" option which lists the names of the files in a given directory without any other information. Direct the output of the "list" command to an output file using output redirection. The resulting output file should then be loaded into EMACS. The "list" command sometimes adds a header line and always adds a line to the end of the list, these extra lines should be eliminated from the text file and the file should be saved back to disk. It should be noted that the MS2SYMXREF utility does not handle file names with embedded spaces. The second parameter <OutputFile> designates the file to which output of the cross reference utility should be sent.

The following is an example of how a file list can be created using the AmigaDOS "list" command. Be sure to edit the file as described above before using it as a file list for the MS2SYMXREF utility.

```
list >ram:liblist 071:MSL quick
```

The output file created by MS2SYMXREF contains a line for each identifier with the following information on each line.

```
<Identifier> <ModuleName> <ObjectClass>
```

```
<Identifier>      - Identifier  
<ModuleName>     - Definition appears in this module  
<ObjectClass>    - CONST, TYPE, VAR, PROCEDURE
```

The following is a short sample of the output from the MS2SYMXREF utility.

```
Lookup      Filesystem  
Close       Filesystem  
PROCEDURE  
PROCEDURE
```

```
Examples:  
MS2SYMXREF liblist.txt  
Sort.txt |> sort.sorted  
          :  
          : Generate sort
```

Chapter 12	Object Module Generator Utility.....	3
	Running the Object Module Generator.....	3

Chapter 12 Object Module Generator Utility
Running the Object Module Generator

This chapter describes a utility included in this package which converts an Amiga load file into an object module acceptable to the Modula-2 linker. The primary use of this utility is to allow modules to be written in assembly language. For additional information refer to the section of this manual which describes how to interface to assembly language.

Running the Object Module Generator

The name of the object module generator program is "M2GenOBM". The command line syntax is as follows:

```
M2GenOBM [Options] <InputFileName> [OutputFileName]
```

The <InputFileName> specifies the root for the name of the files to be read. The conversion program adds the extension ".SBM" to read the symbol file and the extension ".COD" to read the load file. The [OutputFileName] parameter may optionally be specified to force the output file to be sent to a different directory. For example:

```
M2GenOBM RAM:StringsLib
Input Symbol File: RAM:StringsLib.SBM
Input Load File: RAM:StringsLib.COD
Output Obj File: RAM:StringsLib.OBM
```

```
M2GenOBM IOPort DF1:IOPort.OBM
Input Symbol File: IOPort.SBM
Input Load File: IOPort.COD
Output Obj File: DF1:IOPort.OBM
```

The following command line options may be specified. A command line option begins with the character "-" followed by a letter. The following is a list of the available options.

* -m

Ignore modules imported by the definition module. Using this option will create a smaller object file and will not force the linker to automatically link in unnecessary modules.

The object file conversion program displays status information about the module being converted. The following is a typical example of the information which may be displayed.

```
Input Symbol File      = Tasks.SBM
Input Amiga Load File  = Tasks.COD
Output Modula-2 Object File = Tasks.OBM
```

Generating Module:

```
Key = 353 123 091 Name = Tasks
```

Modules Imported by DEFINITION:

```
Key = 234 655 566 Name = Nodes
```

```
Key = 545 767 123 Name = Lists
```

```
DEFINITION Procedures = 10
```

```
DEFINITION Modules    = 2
```

```
DEFINITION Variable Size = 16
```

```
IMPLEMENTATION Variable Size = 12
```

```
IMPLEMENTATION Code Size = 122
```

Description:

Generating Module

The name and key of the module being generated.

Modules Imported by DEFINITION

The list of modules imported by the definition module.

DEFINITION Procedures

Number of procedures defined in the definition module.

DEFINITION Modules

Number of modules imported by the definition module.

DEFINITION Variable Size

Total size of variables defined in the definition module.

IMPLEMENTATION Variable Size

Total size of variables defined in the implementation module.

IMPLEMENTATION Code Size

Total size of code defined in the implementation module.

Chapter 13	Assembly Language Interface.....	3
	Background Information.....	3
	Compiling the Definition Module.....	3
	Assembling the Implementation Module.....	3
	Linking the Object File.....	4
	Converting to an Object Module.....	4
	Example Conversion Batch File.....	5
	Example of an Assembly Language Module.....	5

Chapter 12 Assembly Language Interface.....	3
Background Information.....	3
Compiling the Definition Module.....	3
Assembling the Implementation Module.....	3
Linking the Object File.....	4
Converting to an Object Module.....	4
Example Conversion Batch File.....	5
Example of an Assembly Language Module.....	5

This chapter describes how to write modules in assembly language which may be called by other modules written in Modula-2. In some circumstances it may be desirable to implement a module of a program in assembly language rather than Modula-2. If the amount of assembly language code is very short it may be practical to use the `INLINE`, `REG` and `SETREG` procedures defined in the `SYSTEM` module to implement the assembly language code. For assembly language subroutines of any significant size the above mentioned facility is not very practical, instead the full assembly language interface described in this chapter should be used.

Background Information

The technique of linking used by Modula-2 is sufficiently different from the schemes used for languages such as "C", PASCAL and FORTRAN that a different format had to be used to represent object files accepted by a linker. This object format is not directly compatible with the object format used by Amiga Assembler and the above mentioned languages. Since this package does not include an assembler the Amiga Assembler must be used to write modules in assembly language. A conversion program is then used to convert from the Amiga format to a format acceptable to the Modula-2 linker. The following steps are required to create an assembly language module:

- * Compiling the definition part of the module with the compiler
- * Assembling the implementation part with the assembler
- * Linking the Amiga format object file into a load file
- * Converting the load file to a Modula-2 object module

The following is a detailed description of each of the steps which must be performed.

Compiling the Definition Module

An assembly language module must have a definition part which is written in Modula-2. This allows any client module to use the module as if it were a normal Modula-2 library module. This technique is employed by many of the library modules supplied with this system. The definition module may contain constants, types, variables and procedure headings.

The definition module is processed by the compiler and a symbol module (`.SBM`) is generated. This file will be imported by the clients of this module, it is also read in by the object file converter program.

Assembling the Implementation Module

The assembly language file is written with the help of the special macros defined in the include file `"M2Lib.i"` and in some cases the include file `"System_Map.i"`. In order to construct a valid Modula-2 object module the macros defined in `"M2Lib.i"` must be utilized. The macros automatically build the header and the tables needed for a valid object file. Refer to the example

assembly language module listed at the end of this chapter for the usage of the macros. One very useful function of the macros is to generate labels for local variables and to alert the user of any errors at assembly time.

The assembly language source must be assembled by an assembler which will generate an ALink compatible object file. The file may contain external references to Amiga library offset labels, but it must not contain external references to any functions. Another words the file must be self contained with the exception of the library offset labels, i.e. `_LVO`.

Linking the Object File

The object file generated by the assembler must be linked, even if it does not contain any external references. If the link operation is successful a load file will be generated by the linker. This load file must be in a format acceptable to the object file converter program included with this package.

Converting to an Object Module

This is the last step in creating a Modula-2 compatible object module from an assembly language file. The conversion from the above mentioned load file to an object module is done using the program "M2GenOBM" included in the package. For additional information refer to the chapter of the manual which describes the "M2GenOBM" program. The program requires the following input files.

- * Load file generated by the Amiga linker (extension .COD)
- * Symbol file generated by the Modula-2 compiler (extension .SBM)

As output the program produces an object module file with the extension ".OBM". If the operation is successful the object module generated may later be linked with other modules by the Modula-2 linker to produce an executable program.

Example Conversion Batch File

The following is an example of a batch file which may be used to automatically generate a symbol and object module from a definition module and an assembly language implementation module.

MakeAsm:

```
.Key module/a
M2 <module>.def ; Create symbol module
Asm <module>.asm -o <module>.o ; Create obj file
ALink <module>.o LIB Amiga.lib TO <module>.cod ; Create load file
Delete <module>.o ; Delete obj file
M2GenOBM -m <module> ; Create object module
Delete <module>.cod ; Delete load file
```

Example of usage: Execute MakeAsm XText

Example of an Assembly Language Module

This subsection contains an example of a module written in assembly language. The example shown is actually the library module "Exec" included in this package. The major difference between this example and user created assembly language programs will be that a user module will have more code, this module makes calls to the Amiga's ROM Kernel.

Modula-2 Definition Module: Exec.def

```
DEFINITION MODULE Exec;
```

```
FROM SYSTEM IMPORT ADDRESS;
```

```
CONST
```

```
  ExecName = "exec.library";
```

```
PROCEDURE Debug;
```

```
PROCEDURE GetCC(): BITSET;
```

```
PROCEDURE RawDoFmt(FormatString: ADDRESS; DataStream: ADDRESS;
  PutChProc: ADDRESS; PutChData: ADDRESS);
```

```
PROCEDURE SetSR(newSR, mask: BITSET): BITSET;
```

```
PROCEDURE SumKickData;
```

```
END Exec.
```


Assemble Language Implementation Module: Exec.asm

```
INCLUDE 'M2/M2Lib.i'
```

```
REFLIB      Debug
REFLIB      GetCC
REFLIB      RawDoFmt
REFLIB      SetSR
REFLIB      SumKickData
```

```
DEFINITION_PROCEDURES 5
DEF_PROC    MODULE_BODY
DEF_PROC    Debug
DEF_PROC    GetCC
DEF_PROC    RawDoFmt
DEF_PROC    SetSR
DEF_PROC    SumKickData
```

```
DEFINITION_MODULES 0
```

```
DEFINITION_VARIABLES
```

```
IMPLEMENTATION_VARIABLES
```

```
START_CODE
```

```
-----
PARAMETER_LIST 4,0
END_PARAMETER_LIST
```

```
PROCEDURE_CODE MODULE_BODY
```

```
EMPTY_MODULE_BODY
```

```
-----
PARAMETER_LIST 4,0
END_PARAMETER_LIST
```

```
PROCEDURE_CODE Debug
move.L AbsExecBase,A6
JUMP Debug
```

```
-----
PARAMETER_LIST 4,0
END_PARAMETER_LIST
```

```
PROCEDURE_CODE GetCC
move.L AbsExecBase,A6
JUMP GetCC
```

```
-----
PARAMETER_LIST 4,16
VAR RawDoFmt_FormatString,ADDRESS
VAR RawDoFmt_DataStream,ADDRESS
VAR RawDoFmt_PutChProc,ADDRESS
```

```
VAR RawDoFmt_PutChData,ADDRESS
END_PARAMETER_LIST
```

```
PROCEDURE_CODE RawDoFmt
move.L RawDoFmt_FormatString(A7),A0
move.L RawDoFmt_DataStream(A7),A1
move.L RawDoFmt_PutChProc(A7),A2
move.L RawDoFmt_PutChData(A7),A3
move.L AbsExecBase,A6
JUMP RawDoFmt
```

```
-----
PARAMETER_LIST 4,4
VAR SetSR_newSR,BITSET
VAR SetSR_mask,BITSET
END_PARAMETER_LIST
```

```
PROCEDURE_CODE SetSR
move.W SetSR_newSR(A7),D0
move.W SetSR_mask(A7),D1
move.L AbsExecBase,A6
JUMP SetSR
```

```
-----
PARAMETER_LIST 4,0
END_PARAMETER_LIST
```

```
PROCEDURE_CODE SumKickData
move.L AbsExecBase,A6
JUMP SumKickData
```

```
-----
END_CODE
```

```
END
```

VAR RawDataPutData, ADDRESS
END_PARAMETER_LIST

PROCEDURE CODE RawDataPut
move.l RawDataPutData, A0
move.l RawDataPutData, A1
move.l RawDataPutData, A2
move.l RawDataPutData, A3
move.l AddressBase, A6
JMP RawDataPut

PARAMETER LIST 4, 4
VAR SetNewR, BITSET
VAR SetNewW, BITSET
END_PARAMETER_LIST

PROCEDURE CODE SetNew
move.w SetNewR, A7, D0
move.w SetNewW, A7, D1
move.l AddressBase, A6
JMP SetNew

PARAMETER LIST 4, 0
END_PARAMETER_LIST

PROCEDURE CODE SumKData
move.l AddressBase, A6
JMP SumKData

END_CODE
END

Chapter 14	Module ASCII	3
	CharIsASCII	4
	CharIsControl	4
	CharIsPrintable	4

Chapter 14 Module ASCII.....	14-2
Chapter 15 Module ASCII.....	14-2
Chapter 16 Module ASCII.....	14-2
Chapter 17 Module ASCII.....	14-2

The module **ASCII** is available in many Modula-2 implementations and contains constants which represent the non-printable ASCII codes. In addition, the module contains some useful procedures for determining if a character is an ASCII, control or printable character.

SYNOPSIS
 CharIsASCII(Char: CHAR): BOOLEAN;
 CharIsControl(Char: CHAR): BOOLEAN;
 CharIsPrintable(Char: CHAR): BOOLEAN;

Char - character to be tested

result - TRUE if satisfies test condition
 FALSE if fails test condition

DESCRIPTION
 The above listed procedures return TRUE if a character is of a specific type or FALSE if a character is of a different type.

EXAMPLES
 The following example scans a string until it encounters a non ASCII character or the end of the string.

```

VAR
  str : ARRAY [0..127] OF CHAR;
  i : CARDINAL;
BEGIN
  i := 0;
  WHILE (str[i] < 32) AND (CharIsASCII(str[i])) DO
    INC(i);
  END;

```


Modula-2 Software Construction Set Reference Guide

NAME

CharIsASCII, CharIsControl, CharIsPrintable - determine character type

SYNOPSIS

```
CharIsASCII(Ch: CHAR): BOOLEAN;      is Ch an ascii character
CharIsControl(Ch: CHAR): BOOLEAN;     is Ch a control character
CharIsPrintable(Ch: CHAR): BOOLEAN;   is Ch a printable character
```

Ch - character to be tested

result - TRUE if satisfies test condition
 FALSE if fails test condition

DESCRIPTION

The above listed procedures return TRUE if a character is of a specific type or FALSE if a character is of a different type.

EXAMPLES

The following example scans a string until it encounters a non ASCII character or the end of the string.

```
VAR
  str : ARRAY [0..127] OF CHAR;
  i   : CARDINAL;
BEGIN
  ...
  i := 0;
  WHILE (str[i] # 0C) AND (CharIsASCII(str[i])) DO
    INC(i);
  END;
  ...
```

Chapter 15	Module Conversions.....	3
	ConvStringToNumber.....	4
	ConvNumberToString.....	5

Chapter 15 Module Conversions	15-2
Converting to Modula-2	15-3
Converting from Modula-2	15-4

The **Conversions** module provides functions for binary to ASCII and ASCII to binary conversions. The functions in this module operate on strings and are therefore not bound to any specific I/O device. For maximum flexibility the function in this module support signed or unsigned numbers of any base.

DESCRIPTION
Converts a string which is expected to contain a valid number into its binary equivalent. Leading spaces are allowed and will be skipped. If the number is specified to be unsigned and a '-' character is found the number is considered invalid. The base of the number determines the type of number which will be allowed, for example:

BASE	TYPE
2	binary
8	octal
10	decimal
16	hexadecimal

The converted binary number will be stored into the num parameter, but only if the number is valid.

EXAMPLES

```

VAR
  s : ARRAY [0..32] OF CHAR; value : LONGWORD;
BEGIN
  ReadString(s);
  IF NOT ConvertingNumber(s, value, FALSE, 10) THEN
    WriteString('Invalid Number Entered');
  END;

```

Modula-2 Software Construction Set Reference Guide

NAME

ConvStringToNumber - ASCII to binary conversion

SYNOPSIS

```
ConvStringToNumber(string: ARRAY OF CHAR;  
                   VAR num: LONGWORD;  
                   signed: BOOLEAN;  
                   base: CARDINAL): BOOLEAN
```

string - string containing a number

num - variable to store binary number into (any 32-bit type)

signed - TRUE = signed number, FALSE = unsigned number

base - the base of the number to be parsed

result - TRUE if valid number, FALSE if invalid number

DESCRIPTION

Converts a string which is expected to contain a valid number into its binary equivalent. Leading spaces are allowed and will be skipped. If the number is specified to be unsigned and a "-" character is found the number is considered invalid. The base of the number determines the type of number which will be allowed, for example:

BASE	TYPE
----	----
2	binary
8	octal
10	decimal
16	hexadecimal

The converted binary number will be stored into the num parameter, but only if the number is valid.

EXAMPLES

```
VAR  
  s : ARRAY [0..39] OF CHAR; value : LONGCARD;  
BEGIN  
  ReadString(s);  
  IF NOT ConvStringToNumber(s, value, FALSE, 10) THEN  
    WriteString("Invalid Number Entered");  
  END;  
  ...
```

NAME

ConvNumberToString - binary to ASCII conversion

SYNOPSIS

```
ConvNumberToString(VAR string: ARRAY OF CHAR;
                   num: LONGWORD;
                   signed: BOOLEAN;
                   base: CARDINAL;
                   digits: CARDINAL;
                   padChar: CHAR);
```

string - buffer large enough to hold the output string

num - number to be converted (any 32-bit type)

signed - TRUE = high bit represents sign of number

FALSE = unsigned number

base - base of number being converted

digits - the minimum number of digits to output

padChar - leading character, usually "0" or " "

DESCRIPTION

Convert a binary number into a string. The resulting string is stored into the parameter **string**. The actual string must be large enough to accommodate the generated string. The **digits** parameter determines the minimum length of the output string. The ASCII number is right justified in the string, the remaining characters are filled with the **padChar**. If the number being converted is specified as signed, and its high-bit is 1 then a "-" character will be output. As with the **ConvStringToNumber** procedure the **base** of the number can be changed to display the number in a different base, for example:

BASE	TYPE
----	----
2	binary
8	octal
10	decimal
16	hexadecimal

EXAMPLES

The following example demonstrates how the binary to ASCII conversion function may be used. This example converts a value into a hexadecimal number with a width of 8 and padded with the character "0".

VAR

```
  outStr : ARRAY [0..39] OF CHAR;
```

BEGIN

```
  ConvNumberToString(outStr, value, FALSE, 16, 8, "0");
```

```
  WriteString(outStr);
```

```
  ...
```

Chapter 16	Module FileSystem.....	3
	Close.....	4
	Delete.....	5
	Lookup.....	6
	SetPos.....	7
	GetPos.....	8
	Length.....	9
	ReadWord.....	10
	WriteWord.....	11
	ReadChar.....	12
	WriteChar.....	13

Chapter 16	Module Filesystem	16-1
Chapter 17	Class	17-1
Chapter 18	Define	18-1
Chapter 19	Lookup	19-1
Chapter 20	SetPos	20-1
Chapter 21	GetPos	21-1
Chapter 22	Length	22-1
Chapter 23	ReadWord	23-1
Chapter 24	WriteWord	24-1
Chapter 25	ReadChar	25-1
Chapter 26	WriteChar	26-1

The standard module **FileSystem** is available in most implementations of Modula-2. This module provides procedures for handling low level I/O. These procedures handle I/O very efficiently by buffering both read and write operations. In addition this module sets no limit on the number of files which may be open simultaneously, except system memory constraints.

Most of the procedures in this module take a file structure as the first parameter. The file structure contains all of the information to keep track of an open file. Only the procedures in the **FileSystem** module are allowed to change the values in any of the fields of the file structure. Some of the fields in the File structure may be read by the user. The File structure is defined as follows:

```

TYPE
  Response = (done, notdone, nomemory)
  File = RECORD
    res : Response; (* read only *)
    eof : BOOLEAN;  (* read only *)
    err : LONGINT;  (* read only *)
    ...
  END;
```

res - is an enumeration which is set to **done**, **notdone** or **nomemory**. The **nomemory** response can only occur after a **Lookup** operation. After all other operations the **res** field will contain **done** meaning the operation was successful or **notdone** meaning the operation failed.

eof - is a flag which may be set to **TRUE** after a read operation which encountered the end of a file.

err - is used to store the error code returned by the **AmigaDOS IoErr** function, this field may be checked when an operation has failed. This field is only valid when the **res** field is set to **notdone**.

The rest of the fields in the **File** structure are considered private and should not be read or written.

The **FileSystem** module also defines a variable **BufferSize**. This variable is read by the **Lookup** procedure to determine how much buffer space to allocate for the file. By default this variable is set to 1024, this has been found to produce good disk I/O performance. The value may be changed by a user program before performing a **Lookup** call. Each open file may have a different buffer size specified. For I/O intensive applications it may be useful to experiment with the **BufferSize** to find a good level of performance.

To avoid loosing data or corrupting the data on a disk, all files opened by a program should be explicitly closed by the program. The Modula-2 run-time support functions will not automatically close a program's files upon exit.

Modula-2 Software Construction Set Reference Guide

NAME

Close - close a currently open file

SYNOPSIS

Close(VAR f: File);

f - File structure

DESCRIPTION

Close a currently open file. If the file's buffer has been changed it will be written to disk automatically. After the file is closed the File variable may be reused for opening another file.

EXAMPLES

This example closes a file to which data was being output. The `res` field of the `File` structure is checked to make sure that no errors occurred when the disk buffer was written to disk.

```
VAR out : File;
```

```
BEGIN
```

```
...  
  Close(out);
```

```
  IF out.res # done THEN
```

```
    WriteString("Output File Error!");
```

```
  END;
```

```
...
```

NAME

Delete - delete a file from the file system

SYNOPSIS

Delete(filename: ARRAY OF CHAR);

filename - name of file to be deleted

DESCRIPTION

Deletes a file of a specified filename. If the filename does not exist, nothing happens.

EXAMPLES

BEGIN

```
...Delete("init.dat");
```


Modula-2 Software Construction Set Reference Guide

NAME

Lookup - open an old file or a new file

SYNOPSIS

```
Lookup(VAR f: File; filename: ARRAY OF CHAR; new: BOOLEAN);
```

f - File structure

filename - name of file to open

new - FALSE = open existing file, TRUE = open new file

DESCRIPTION

Attempts to open a file of a specified filename. If the new parameter is TRUE then a new file is created, if a file of the same name already exists it is automatically deleted. If the new parameter is FALSE then an existing file of the same name must already exist. If the specified file could not be opened the res field of the File structure is set to notdone. The Lookup function may fail if not enough memory exists to allocate the disk buffer for this file, in this case the res field of the File structure is set to nomemory.

EXAMPLES

This example attempts to open an existing file and checks if the file has been opened successfully.

```
VAR out : File;
```

```
BEGIN
```

```
  Lookup(out, "test.dat", FALSE);
```

```
  IF out.res # done THEN
```

```
    WriteString(" 'test.dat' data file not opened!");
```

```
  END;
```

```
...
```

NAME

SetPos - move the file pointer to a new position

SYNOPSIS

SetPos(VAR f: File; VAR pos: LONGCARD);

f - File structure

pos - new file pointer position relative to beginning of file

DESCRIPTION

Move the file pointer to a new position. This operation is very efficient if the new position is within the current buffer, otherwise the current buffer will be written to disk and a new buffer loaded. The file pointer can not be moved to a position beyond the end of the file. After the file pointer is moved all read and write operations will be from the new position.

EXAMPLES

In this example the file pointer is moved to the beginning of a file.

VAR inFile : File;

BEGIN

...
SetPos(inFile, 00);
...

Modula-2 Software Construction Set Reference Guide

NAME

GetPos - return the current file pointer position

SYNOPSIS

GetPos(VAR f: File; VAR pos: LONGCARD);

f - File structure

pos - store the current file pointer into this variable

DESCRIPTION

Get the current file pointer position and store it into the parameter pos.

EXAMPLES

The following example moves the file pointer to a new position writes out the character "." and then restores the file pointer to its original position.

```
VAR out : File; oldPos, newPos : LONGCARD;
```

```
BEGIN
```

```
...
  GetPos(out, oldPos);
  SetPos(out, newPos);
  WriteChar(out, ".");
  SetPos(out, oldPos);
...
```

NAME

Length - return current length of file

SYNOPSIS

Length(f: File; VAR length: LONGCARD);

f - File structure

length - store length of file into this variable

DESCRIPTION

Determines the current length of a file and stores it into the parameter **length**. The length is the actual number of bytes in the file. This is not necessarily the same as the amount of disk space taken up by the file, because the operating system rounds the file to the nearest block.

EXAMPLES

VAR in : File; flength : LONGCARD;

BEGIN

...
Length(in, flength);
...

Modula-2 Software Construction Set Reference Guide

NAME

ReadWord - read a word(16-bits) of data from a file

SYNOPSIS

```
ReadWord(VAR f: File; VAR w: WORD);
```

f - File structure

w - variable to store the data in (any 16-bit type)

DESCRIPTION

Read a word of data from a file starting at the current file pointer position. The data is stored into the parameter w, the actual parameter must be a 16-bit variable.

EXAMPLES

```
VAR inFile : File; count : CARDINAL;
```

```
BEGIN
```

```
  ...  
  ReadWord(inFile, count);
```

```
  ...
```

NAME

WriteWord - write a word(16-bits) of data to a file

SYNOPSIS

WriteWord(VAR f: File; w: WORD);

f - File structure

w - a word value to be written

DESCRIPTION

Output a word(16-bits) of data at the current file pointer position. In most cases the data is stored into a buffer and later when the buffer is full the data is written to disk.

EXAMPLES

VAR outFile : File; x, y : CARDINAL;

BEGIN

...
WriteWord(outFile, x);
WriteWord(outFile, y);
...

Modula-2 Software Construction Set Reference Guide

NAME

ReadChar - read a character from a file

SYNOPSIS

```
ReadChar(VAR f: File; VAR ch: CHAR);
```

f - File structure

ch - variable into which the data is to be stored

DESCRIPTION

Read a character of data from a file starting at the current file pointer position. The data is stored into the parameter ch.

EXAMPLES

The following example reads a string from the inFile and stores the string into an array of characters called msg.

```
VAR inFile : File; msg : ARRAY [0..255] OF CHAR; ch : CHAR;
```

```
  i : INTEGER;
```

```
BEGIN
```

```
...
```

```
  i := 0;
```

```
  REPEAT
```

```
    ReadChar(inFile, ch);
```

```
    msg[i] := ch;
```

```
    INC(i);
```

```
  UNTIL (ch = 0C);
```

```
...
```

NAME

WriteChar - write a character to a file

SYNOPSIS

WriteChar(VAR f: File; ch: CHAR);

f - File structure

ch - character to be written to the file

DESCRIPTION

Output a character(8-bits) of data at the current file pointer position. In most cases the data is stored into a buffer and later when the buffer is full the data is written to disk.

EXAMPLES

The following example writes a string to the outFile from an array of characters called msg.

```
VAR outFile : File; msg : ARRAY [0..255] OF CHAR; ch : CHAR;
    i : INTEGER;
BEGIN
    ...
    i := 0;
    REPEAT
        ch := msg[i];
        INC(i);
        WriteChar(inFile, ch);
    UNTIL (ch = 0C);
    ...
```

Chapter 17	Module Heap	3
ALLOCATE		4
DEALLOCATE		5
Available		6
FreeHeap		7

Chapter 17 Module Heap.....	17-2
ALLOCATE.....	17-3
DEALLOCATE.....	17-4
Available.....	17-5
FreeHeap.....	17-6

The module **Heap** is very similar to the standard **Storage** module. The difference between the **Heap** module and the **Storage** module is that the **Heap** module keeps track of the memory allocated and freed. Therefore, it is easy to free all memory currently allocated by calling the **FreeHeap** procedure.

This module allocates memory using the ROM Kernel **AllocMem** function with the memory type **MemReqSet{}**. This memory type tries to allocate fast memory first if available, otherwise chip memory is allocated.

The disadvantage of this module over the standard **Storage** module is that it has an overhead of 12 bytes per memory block allocated. There is no execution speed penalty for using the **Heap** module over the **Storage** module. However, in a program which allocates a very large number of small memory blocks this module would cause a much larger amount of memory to be allocated.

Modula-2 Software Construction Set Reference Guide

NAME

ALLOCATE - allocate an area of a given size

SYNOPSIS

ALLOCATE(VAR a: ADDRESS; size: LONGCARD);

a - store address of memory allocated or NIL if no memory
size - number of bytes to allocate

DESCRIPTION

Allocate a block of memory of a specified **size**. The memory is allocated by calling the ROM Kernel **AllocMem** function with a **MemReqSet{}**. If the memory could not be allocated a **NIL** is returned as the address.

EXAMPLES

```
VAR Table : POINTER TO ARRAY [0..1023] OF BOOLEAN;

BEGIN
  ALLOCATE(Table, 10240);
  IF Table = NIL THEN
    WriteString("insufficient memory to continue!");
  END;
  ...
```

NAME

DEALLOCATE - return memory allocated by ALLOCATE

SYNOPSIS

DEALLOCATE(VAR a: ADDRESS; size: LONGCARD);

a - address of the memory to be deallocated
size - number of bytes to free

DESCRIPTION

Deallocates memory allocated with ALLOCATE. The memory block being freed must be exactly the same as the block returned by ALLOCATE.

EXAMPLES

VAR Table : POINTER TO ARRAY [0..1023] OF BOOLEAN;

BEGIN

ALLOCATE(Table, 10240);

IF Table = NIL THEN

WriteString("insufficient memory to continue!");

END;

...

DEALLOCATE(Table, 10240);

...

Modula-2 Software Construction Set Reference Guide

NAME

Available - determine if a specified amount of memory is free

SYNOPSIS

Available(size: LONGCARD): BOOLEAN;

size - number of bytes that needs to be available

result - TRUE if the specified number of bytes are available

DESCRIPTION

Check if a specified number of bytes are available as contiguous memory. If the memory has become fragmented then this procedure may return FALSE even when the total amount of free memory exceeds the specified size. Modula-2 programs moved from other system may use this function before using the ALLOCATE function.

EXAMPLES

```
VAR Table : ARRAY [0..1999] OF CHAR;
```

```
BEGIN
```

```
...
  IF Available(20000) THEN
    ALLOCATE(Table, 20000);
  END;
...
```

NAME

FreeHeap - free all memory blocks currently allocated

SYNOPSIS

FreeHeap();

DESCRIPTION

Free all memory blocks currently allocated by the caller using the ALLOCATE procedure. This procedure may be useful if a fatal error is encountered and it is necessary to free all allocated memory without going through the normal cleanup procedure.

EXAMPLES

TYPE

```
NodePtr = POINTER TO Node;
Node = RECORD
    Next : NodePtr;
    Prev : NodePtr;
    Name : ARRAY [0..15] OF CHAR;
END;
```

VAR

```
    p, q : NodePtr;
BEGIN
    ...
    ALLOCATE(p, SIZE(p^));
    ...
    ALLOCATE(q, SIZE(q^));
    ...
    FreeHeap();
    ...
```


Chapter 18	Module InitMathLib0.....	3
	OpenMathLib0.....	4
	CloseMathLib0.....	5

Modula-2 Software Construction Set Reference Guide

Chapter 18 Module Initialization	18-2
OpenModule	18-2
CloseModule	18-2

The `InitMathLib0` module performs the necessary initialization which is required before any of the procedure in the `MathLib0` module are used. The `MathLib0` module utilizes the Amiga's "mathtrans.library", this library is not opened automatically since the "mathtrans.library" is disk based and always opening this library would cause an unnecessary overhead on programs.

This module is especially useful when attempting to port programs written on other systems which utilize the module `MathLib0`. By using the open and close procedures in this module a program from another system can be made to work without writing any additional code.

DESCRIPTION
This procedure is used to open the Amiga's "mathtrans.library". If the procedure returns FALSE there are probably two possible reasons the file "mathtrans.library" was not found in the "lib:" directory or there was insufficient memory to load the file from disk. In either case the program should exit gracefully because calling any of the functions in the module MathLib0 would cause a system crash.

EXAMPLES
The following demonstrates how the `OpenMathLib0` procedure may be used.

```

BEGIN
  IF NOT OpenMathLib0 THEN
    WriteString("FATAL ERROR: 'mathtrans.library' not opened!");
    HALT; ("exit");
  END;

```

Modula-2 Software Construction Set Reference Guide

NAME

OpenMathLib0 - open the "mathtrans.library"

SYNOPSIS

OpenMathLib0(): BOOLEAN;

result - TRUE if successfully opened "mathtrans.library"
FALSE if failed to open

DESCRIPTION

This procedure is used to open the Amiga's "mathtrans.library". If the procedure returns FALSE there are probably two possible reasons the file "mathtrans.library" was not found in the "libs:" directory or there was insufficient memory to load the file from disk, in either case the program should exit gracefully because calling any of the functions in the module MathLib0 would cause a system crash.

EXAMPLES

The following demonstrates how the OpenMathLib0 procedure may be used.

BEGIN

```
IF NOT OpenMathLib0() THEN
  WriteString("FATAL ERROR: 'mathtrans.library' not opened!");
  HALT; (* exit *)
END;
...
```

NAME

CloseMathLib0 - close the "mathtrans.library"

SYNOPSIS

CloseMathLib0();

DESCRIPTION

This procedure should be called before exiting the program which uses the module **MathLib0**. This procedure closes the Amiga's "mathtrans.library" and should therefore only be called if the **OpenMathLib0** function was successful. If the **CloseMathLib0** is not performed the library will not be removed from memory until the system is rebooted.

EXAMPLES

The following demonstrates how the **OpenMathLib0** and **CloseMathLib0** procedures may be used.

BEGIN

```
IF NOT OpenMathLib0() THEN
  WriteString("FATAL ERROR: 'mathtrans.library' not opened!");
  HALT; (* exit *)
END;
...
CloseMathLib0();
...
```

Chapter 19	Module InOut	3
	OpenInput	4
	OpenOutput	5
	OpenInputFile	6
	OpenOutputFile	7
	CloseInput	8
	CloseOutput	9
	Read	10
	ReadString	11
	ReadInt	12
	ReadCard	13
	ReadWrd	14
	Write	15
	WriteLn	16
	WriteString	17
	WriteInt	18
	WriteCard	18
	WriteOct	18
	WriteHex	18
	WriteWrd	18

Modula-2 Software Construction Set Reference Guide

Chapter 19 Module Input	19
OpenInput	19
OpenOutput	19
OpenOutputFile	19
OpenOutputFile	19
CloseInput	19
CloseOutput	19
Read	19
ReadString	19
ReadInt	19
ReadCard	19
ReadWord	19
Write	19
WriteInt	19
WriteString	19
WriteInt	19
WriteCard	19
WriteOut	19
WriteHex	19
WriteWord	19

The standard module **InOut** is available in most implementations of Modula-2. The module provides high level I/O operations to an output stream and from an input stream. It is important to understand that the procedures in this module can only work with one input and one output stream at one time.

By default the input stream is connected to the terminal **StdInput** and the output stream is connected to the terminal **StdOutput**. This means that any procedure which performs a write will send data to the users terminal and any procedure which performs a read will get its input from the users terminal. To force data to be written to a file, the **OpenOutput** or **OpenOutputFile** function must be used to establish the connection to the file. The same is true for input, except the **OpenInput** or the **OpenInputFile** function is used. After all input or output operations have been completed on a file, a **CloseInput** or **CloseOutput** function must be called, this will cause the file to be closed and revert the input or output stream back to the terminal.

This module defines a variable **Done** which is set to TRUE or FALSE depending on if the previous call to a function in the module **InOut** was successful or unsuccessful. The variable **termCH** is set to the character which terminated input by the **ReadString**, **ReadInt** and **ReadCard** functions. The variables **in** and **out** are used by the **InOut** module when performing input or output to a file, these variables should not be manipulated as that may disrupt the proper operation of the module. The variable **Echo** controls the operation of the **ReadString** procedure, this variable is non-standard. When **Echo** is set to TRUE **ReadString** will echo any input, when FALSE the input will not be echoed. By default the variable is set to TRUE. The value to be used for **Echo** depends on the type of output window through which the i/o is taking place. If the i/o is going through a "CON:" window, like the one used by the CLI the **Echo** variable should be set to FALSE. When performing i/o through a "RAW:" window, then the **Echo** variable should be set to TRUE.

The module **InOut** defines the constant **EOL**, this represents the ASCII code for the end of line character for this implementation of the **InOut** module.

Modula-2 Software Construction Set Reference Guide

NAME

OpenInput - open a user specified input file

SYNOPSIS

OpenInput(defext: ARRAY OF CHAR);

defext - extension to append if name ends in "."

DESCRIPTION

Prompts the user for the name of a file. The prompt output to the terminal is "in>". The user can abort the prompt by entering ESC, which will return with the variable **Done** set to FALSE. If the user enters a name and the name ends in a period, then the parameter **defext** will be appended to the name of the file. An attempt will be made to open the file. If the file can not be found then the message " not found" will be output to the terminal. The user will then be prompted to enter another name. If the file is successfully opened then the function returns with the variable **Done** set to TRUE.

EXAMPLES

In the following example the user will be prompted to specify an input file. If the user aborts by pressing ESC then the program will be aborted.

BEGIN

```
OpenInput(".txt");
IF NOT Done THEN
  HALT;
END;
...
```

NAME

OpenOutput - open a user specified output file

SYNOPSIS

OpenOutput(defext: ARRAY OF CHAR);

defext - extension to append if the name ends with "."

DESCRIPTION

Prompts the user for the name of a file. The prompt output to the terminal is "out>". The user can abort the prompt by entering ESC, which will return with the variable **Done** set to FALSE. If the user enters a name and the name ends in a period, then the parameter **defext** will be appended to the name of the file. An attempt will be made to open the file. If the file can not be opened for output then the user will then be prompted to enter another name. If the file is successfully opened then the function returns with the variable **Done** set to TRUE.

EXAMPLES

In the following example the user will be prompted to specify an output file. If the user aborts by pressing ESC then the program will be aborted.

BEGIN

OpenOutput(".bak");

IF NOT Done THEN

HALT;

END;

...

Modula-2 Software Construction Set Reference Guide

NAME

OpenInputFile - open a file for input

SYNOPSIS

OpenInputFile(filename: ARRAY OF CHAR);

filename - name of file to open

DESCRIPTION

The specified file is opened for input. If the operation is successful then the variable **Done** is set to TRUE. If the open operation failed then the variable **Done** is set to FALSE.

EXAMPLES

BEGIN

OpenInputFile("help.dat");

IF NOT Done THEN

WriteString("Error: input file not opened!");

HALT;

END;

...

NAME

OpenOutputFile - open a file for output

SYNOPSIS

OpenOutputFile(filename: ARRAY OF CHAR);

filename - name of file to open

DESCRIPTION

The specified file is opened for output. If the operation is successful then the variable **Done** is set to TRUE. If the open operation failed then the variable **Done** is set to FALSE.

EXAMPLES

BEGIN

OpenOutputFile("results.dat");

IF NOT Done THEN

WriteString("Error: output file not opened!");

HALT;

END;

...

Modula-2 Software Construction Set Reference Guide

NAME

CloseInput - close the input file

SYNOPSIS

CloseInput();

DESCRIPTION

Closes the file from which input is currently being received and returns to receiving input from the terminal. This procedure is used to close a file opened with **OpenInput** or **OpenInputFile**.

EXAMPLES

BEGIN

OpenInput(".tmp");

...

CloseInput();

...

NAME

CloseOutput - close the output file

SYNOPSIS

CloseOutput();

DESCRIPTION

Close the file to which the output is currently being sent and begin sending output to the terminal. This procedure is used to close a file opened with **OpenOutput** or **OpenOutputFile**.

EXAMPLES

BEGIN

OpenOutput(".tmp");

...

CloseOutput();

...

Modula-2 Software Construction Set Reference Guide

NAME

Read - read a char from the input stream

SYNOPSIS

Read(VAR ch: CHAR);

ch - variable to store the next character in

DESCRIPTION

Read the next character from the input stream and store it into the parameter ch. Set the variable Done to TRUE if the operation was successful. If an end of file condition was detected set Done to FALSE.

EXAMPLES

```
VAR cmdCh : CHAR;
```

```
BEGIN
```

```
...
```

```
  LOOP
```

```
    Read(cmdCh);
```

```
    IF NOT Done THEN
```

```
      EXIT;
```

```
    END;
```

```
  ...
```

```
END;
```

```
...
```

NAME

ReadString - read a string from the input stream

SYNOPSIS

ReadString(VAR s: ARRAY OF CHAR);

s - array which is to receive the input string

DESCRIPTION

Read a sequence of characters not containing blank or control characters. Leading blanks are ignored. Input is terminated by a carriage return or blank space. To delete a character the backspace key can be used. The character which caused the input to be terminated is stored in termCH. The state of the variable Echo determines if this procedure will echo the characters typed by the user.

EXAMPLES

The following example prompts the user for a command, then reads the command from the input stream.

VAR cmd : ARRAY [0..127] OF CHAR;

BEGIN

...
 WriteString("Enter next Command? ");
 ReadString(cmd);
 ...

Modula-2 Software Construction Set Reference Guide

NAME

ReadInt - read an integer and convert to binary

SYNOPSIS

```
ReadInt(VAR x: INTEGER);
```

x - integer variable into which to store the binary value

DESCRIPTION

Read a string from the input stream. The string should contain an ASCII integer. Convert the integer to binary and store into the parameter x. The variable **Done** is set to TRUE if the integer is read or FALSE if the operation failed.

The syntax for an integer is:

```
integer = ["+"|"-"] digit {digit}.
```

EXAMPLES

```
VAR x : INTEGER;
```

```
BEGIN
```

```
...
  WriteString("Enter the x coordinate? ");
  ReadInt(x);
  IF NOT Done THEN
    WriteString("Invalid x coordinate!");
  END;
...
```

NAME

ReadCard - read a cardinal and convert to binary

SYNOPSIS

ReadCard(VAR x: CARDINAL);

x - cardinal variable into which to store the binary value

DESCRIPTION

Read a string from the input stream. The string should contain an ASCII cardinal. Convert the cardinal to binary and store into the parameter x. The variable **Done** is set to TRUE if the cardinal is read or FALSE if the operation failed.

The syntax for a cardinal is:

cardinal = digit {digit}.

EXAMPLES

VAR age : CARDINAL;

BEGIN

```
...
  WriteString("Enter your age? ");
  ReadCard(age);
  IF NOT Done THEN
    WriteString("Invalid Age!");
  END;
...
```

Modula-2 Software Construction Set Reference Guide

NAME

ReadWrd - read a word(16-bits) of data from the input stream

SYNOPSIS

```
ReadWrd(VAR w: WORD);
```

w - variable into which to store the word value

DESCRIPTION

Read a word(16-bits) of data into the parameter w from the input stream. The variable Done is set to TRUE if the operation is successful. If an end of file condition is encountered the variable Done is set to FALSE.

EXAMPLES

The following example reads from the input stream until an end of file is reached.

```
VAR val : CARDINAL;
```

```
BEGIN
```

```
...
```

```
  LOOP
```

```
    ReadWrd(val);
```

```
    IF NOT Done THEN
```

```
      EXIT;
```

```
    END;
```

```
  END;
```

```
...
```

NAME

Write - write a character to the output stream

SYNOPSIS

```
Write(ch: CHAR);
```

ch - the character to be sent to the output stream

DESCRIPTION

Output a character to the output stream.

EXAMPLES

The following example outputs the string "12" to the output stream.

BEGIN

```
...  
  Write("1");  
  Write("2");  
...
```


Modula-2 Software Construction Set Reference Guide

NAME

WriteLn - terminate the current line of the output stream

SYNOPSIS

WriteLn;

DESCRIPTION

Terminate the current output line by sending a line feed to the output stream.

EXAMPLES

The following example will cause "One..." to be sent to the output stream and "Two..." to be sent to the next line.

BEGIN

```
...  
  WriteString("One...");  
  WriteLn;  
  WriteString("Two...");  
  WriteLn;  
...
```

NAME

WriteString - output a string to the output stream

SYNOPSIS

WriteString(s: ARRAY OF CHAR);

s - string to be output to the output stream

DESCRIPTION

Output a string to the output stream. The characters of the string are output until a null character is encountered. No line feed is generated after the string is output.

EXAMPLES

BEGIN

... WriteString("Hello World!");

Modula-2 Software Construction Set Reference Guide

NAME

WriteInt, WriteCard, WriteOct, WriteHex - write a number to output stream

SYNOPSIS

```
WriteInt(x,n: INTEGER);      write x in decimal to output stream
WriteCard(x,n: CARDINAL);    write x in decimal to output stream
WriteOct(x,n: CARDINAL);     write x in octal to output stream
WriteHex(x,n: CARDINAL);     write x in hexadecimal to output stream
```

x - integer or cardinal binary number to be output

n - minimum number of characters to output

DESCRIPTION

Output an ASCII representation of an integer or cardinal number. The output is in decimal, octal or hexadecimal. The parameter n specifies the minimum width of the output string. If the number of digits is less than the minimum specified then the field is padded with space characters. The WriteHex procedure always outputs 4 digits, this is done to make hexadecimal numbers easily distinguishable. The output is sent to the output stream.

EXAMPLES

BEGIN

```
...
WriteHex(1024, 4);
WriteLn;
WriteCard(129, 5);
...
```

Chapter 20	Module LargeSets	3
CreateSet		4
DeleteSet		5
SetInfo		6
SetRange		7
In		8
InclElement		9
ExclElement		10
InclRange		11
ExclRange		12
CopySet		13
Union		14
Diff		15
Intersection		16
SymmetricDiff		17

Chapter 20 Module Libraries	20
Chapter 21 Module Libraries	21
Chapter 22 Module Libraries	22
Chapter 23 Module Libraries	23
Chapter 24 Module Libraries	24
Chapter 25 Module Libraries	25
Chapter 26 Module Libraries	26
Chapter 27 Module Libraries	27
Chapter 28 Module Libraries	28
Chapter 29 Module Libraries	29
Chapter 30 Module Libraries	30
Chapter 31 Module Libraries	31
Chapter 32 Module Libraries	32
Chapter 33 Module Libraries	33
Chapter 34 Module Libraries	34
Chapter 35 Module Libraries	35
Chapter 36 Module Libraries	36
Chapter 37 Module Libraries	37
Chapter 38 Module Libraries	38
Chapter 39 Module Libraries	39
Chapter 40 Module Libraries	40
Chapter 41 Module Libraries	41
Chapter 42 Module Libraries	42
Chapter 43 Module Libraries	43
Chapter 44 Module Libraries	44
Chapter 45 Module Libraries	45
Chapter 46 Module Libraries	46
Chapter 47 Module Libraries	47
Chapter 48 Module Libraries	48
Chapter 49 Module Libraries	49
Chapter 50 Module Libraries	50
Chapter 51 Module Libraries	51
Chapter 52 Module Libraries	52
Chapter 53 Module Libraries	53
Chapter 54 Module Libraries	54
Chapter 55 Module Libraries	55
Chapter 56 Module Libraries	56
Chapter 57 Module Libraries	57
Chapter 58 Module Libraries	58
Chapter 59 Module Libraries	59
Chapter 60 Module Libraries	60
Chapter 61 Module Libraries	61
Chapter 62 Module Libraries	62
Chapter 63 Module Libraries	63
Chapter 64 Module Libraries	64
Chapter 65 Module Libraries	65
Chapter 66 Module Libraries	66
Chapter 67 Module Libraries	67
Chapter 68 Module Libraries	68
Chapter 69 Module Libraries	69
Chapter 70 Module Libraries	70
Chapter 71 Module Libraries	71
Chapter 72 Module Libraries	72
Chapter 73 Module Libraries	73
Chapter 74 Module Libraries	74
Chapter 75 Module Libraries	75
Chapter 76 Module Libraries	76
Chapter 77 Module Libraries	77
Chapter 78 Module Libraries	78
Chapter 79 Module Libraries	79
Chapter 80 Module Libraries	80
Chapter 81 Module Libraries	81
Chapter 82 Module Libraries	82
Chapter 83 Module Libraries	83
Chapter 84 Module Libraries	84
Chapter 85 Module Libraries	85
Chapter 86 Module Libraries	86
Chapter 87 Module Libraries	87
Chapter 88 Module Libraries	88
Chapter 89 Module Libraries	89
Chapter 90 Module Libraries	90
Chapter 91 Module Libraries	91
Chapter 92 Module Libraries	92
Chapter 93 Module Libraries	93
Chapter 94 Module Libraries	94
Chapter 95 Module Libraries	95
Chapter 96 Module Libraries	96
Chapter 97 Module Libraries	97
Chapter 98 Module Libraries	98
Chapter 99 Module Libraries	99
Chapter 100 Module Libraries	100

The **LargeSets** module implements machine independent large sets. Modula-2 specifies that the set type must not be larger than the machine word. For this implementation the set can have up to 32 elements on smaller computers the set may be limited to as few as 16 elements. The **LargeSets** module can be used to create a set with as many as 65536 elements.

The program refers to a set by its handle. The handle is defined as a hidden type called **LargeSet**. To create the actual set the procedure **CreateSet** is called with the range of the set to be created. Once a set is created any number of operations can be performed such as adding elements to the set, testing if an element exists in the set and many other useful operations. When the program no longer needs the set it should be deleted using **DeleteSet**.

Operations between two sets are only possible when the two sets have exactly the same range. Further, it is not legal to include, exclude or test any element which is outside the range of the set.

The amount of memory required to represent a large set not including the small set management overhead is calculated as follows:

$$\text{SetSize} = ((\text{lastElement} - \text{firstElement}) \text{ DIV } 8) + 1;$$

Modula-2 Software Construction Set Reference Guide

NAME

CreateSet - allocate memory for a large set

SYNOPSIS

CreateSet(VAR set: LargeSet; first, last: CARDINAL): BOOLEAN;

set - variable to store handle for LargeSet

first - first element in range of set

last - last element in range of set

result - TRUE if allocated successfully

FALSE if set allocation failed

DESCRIPTION

Allocate memory for storage of a large set with the specified range. The set is initially set to empty. If the result returned is FALSE then the set was not created because of insufficient memory.

EXAMPLES

```
VAR
  MySet : LargeSet;
BEGIN
  IF CreateSet(MySet, 0, 999) THEN
    WriteString("CreateSet() Successful!");
  ELSE
    WriteString("CreateSet() Failed!");
  END;
  ...
```


NAME

DeleteSet - deallocate a large set

SYNOPSIS

DeleteSet(set: LargeSet);

set - handle of LargeSet to be deleted

DESCRIPTION

This procedure releases the dynamic memory occupied by a large set. This operation is performed after the set is no longer needed. This operation should only be performed on a set that was successfully allocated.

EXAMPLES

VAR

MySet : LargeSet;

BEGIN

IF CreateSet(MySet, 0, 999) THEN

 DeleteSet(MySet);

END;

...

Modula-2 Software Construction Set Reference Guide

NAME

SetInfo - return internal information about a large set

SYNOPSIS

```
SetInfo(set: LargeSet; VAR setAdr: ADDRESS; VAR setSize: CARDINAL);
```

set - handle of LargeSet

setAdr - variable to store address of first byte of LargeSet

setSize - variable to store size of LargeSet in bytes

DESCRIPTION

This procedure returns private information about the storage for a large set. This procedure should be used with care, it is mainly provided so that the elements of a large set can be read from or written to a file on a disk.

EXAMPLES

This example shows a procedure which might be used to write the contents of a set to a file.

```
...
PROCEDURE SaveSet(file: FileHandle; set: LargeSet);
VAR
  sAdr : ADDRESS;
  sSize: CARDINAL;
  err  : LONGINT;
BEGIN
  SetInfo(set, sAdr, sSize);
  err := Write(file, sAdr, sSize);
END SaveSet;
...
```

NAME

SetRange - return the element range of a large set

SYNOPSIS

```
SetRange(set: LargeSet; VAR first, last: CARDINAL);
```

set - handle of LargeSet

first - variable to store first element of set

last - variable to store last element of set

DESCRIPTION

This procedure returns to the caller the first and last element of the set range. The range returned is exactly the same as the range specified when the set was created using the CreateSet procedure.

EXAMPLES

The following example implements a procedure which displays on the screen the range of a specified set.

```
...
PROCEDURE DispSetSize(set: LargeSet);
VAR
  first, last: CARDINAL;
BEGIN
  SetRange(set, first, last);
  argo[0].W := first;
  argo[1].W := last;
  printf("[%u..%u]", argo);
END DispSetSize;
...
```

Modula-2 Software Construction Set Reference Guide

NAME

In - test if an element is present in a large set

SYNOPSIS

In(set: LargeSet; element: CARDINAL): BOOLEAN;

set - handle of LargeSet

element - element to test

result - TRUE if element in set

FALSE if element not in set

DESCRIPTION

This procedure tests if a specified element exists in a large set. The specified element must lie in the range of the set for the result returned to be considered valid. This procedure corresponds directly to the standard 'IN' operator used by Modula-2.

EXAMPLES

```
VAR
  MySet : LargeSet;
BEGIN
  IF CreateSet(MySet, 0, 999) THEN
    ...
    IF In(MySet, 5) THEN
      WriteString("Element Found In Set!");
    ELSE
      WriteString("Element Not Found In Set!");
    END;
    ...
    DeleteSet(MySet);
  END;
  ...
```

NAME

InclElement - include an element in a large set

SYNOPSIS

InclElement(set: LargeSet; element: CARDINAL);

set - handle of LargeSet
element - element to include

DESCRIPTION

Include a specified element in a large set. The specified element must lie in the range of the set for the operation to be considered valid. This procedure corresponds directly to the standard 'INCL' operator used by Modula-2.

EXAMPLES

```
VAR
  MySet : LargeSet;
BEGIN
  IF CreateSet(MySet, 0, 999) THEN
    ...
    InclElement(MySet, 100);
    ...
    DeleteSet(MySet);
  END;
  ...
```

Modula-2 Software Construction Set Reference Guide

NAME

ExclElement - exclude an element from a large set

SYNOPSIS

ExclElement(set: LargeSet; element: CARDINAL);

set - handle of LargeSet

element - element to exclude

DESCRIPTION

Exclude a specified element from a large set. The specified element must lie in the range of the set for the operation to be considered valid. This procedure corresponds directly to the standard 'EXCL' operator used by Modula-2.

EXAMPLES

VAR

MySet : LargeSet;

BEGIN

IF CreateSet(MySet, 0, 999) THEN

...
ExclElement(MySet, 45);

...
DeleteSet(MySet);

END;

...

NAME

InclRange - include a range of elements in a large set

SYNOPSIS

InclRange(set: LargeSet; first, last: CARDINAL);

set - handle to LargeSet

first - first element of range to be included

last - last element of range to be included

DESCRIPTION

Include a range of elements in a large set. The specified range must be a subrange of the large set in order for this operation to be valid.

EXAMPLES

VAR

MySet : LargeSet;

BEGIN

IF CreateSet(MySet, 0, 999) THEN

...

InclRange(MySet, 10, 100);

...

DeleteSet(MySet);

END;

...

Modula-2 Software Construction Set Reference Guide

NAME

ExclRange - exclude a range of elements from a large set

SYNOPSIS

ExclRange(set: LargeSet; first, last: CARDINAL);

set - handle to LargeSet

first - first element of range to be excluded

last - last element of range to be excluded

DESCRIPTION

Exclude a range of elements from a large set. The specified range must be a subrange of the large set in order for this operation to be valid.

EXAMPLES

```
VAR
  MySet : LargeSet;
BEGIN
  IF CreateSet(MySet, 0, 999) THEN
    ...
    ExclRange(MySet, 100, 200);
    ...
    DeleteSet(MySet);
  END;
  ...
```

NAME

CopySet - copy the elements of a large set

SYNOPSIS

CopySet(dstSet, srcSet: LargeSet);

dstSet - destination for copy operation

srcSet - source for copy operation

DESCRIPTION

Copy the elements from a source set to a destination set. This operation is only valid if both the source set and the destination set have exactly the same set range.

EXAMPLES

VAR

MySet, YourSet : LargeSet;

BEGIN

IF (CreateSet(MySet, 0, 999)) AND (CreateSet(YourSet, 0, 999)) THEN

...

CopySet(MySet, YourSet);

...

DeleteSet(MySet);

DeleteSet(YourSet);

END;

...

Modula-2 Software Construction Set Reference Guide

NAME

Union - perform a union operation on two large sets

SYNOPSIS

```
Union(dstSet, srcSet: LargeSet);
```

dstSet - destination for union operation

srcSet - source for union operation

DESCRIPTION

Perform a set union operation. The result of the union of the two sets is stored back into the **dstSet**. Both sets must have exactly the same set range. This procedure corresponds directly to the standard Modula-2 '+' set operator.

EXAMPLES

```
VAR
  MySet, YourSet : LargeSet;
BEGIN
  IF (CreateSet(MySet, 0, 999)) AND (CreateSet(YourSet, 0, 999)) THEN
    ...
    Union(MySet, YourSet);
    ...
    DeleteSet(MySet);
    DeleteSet(YourSet);
  END;
  ...
```

NAME

Diff - perform a difference operation on two large sets

SYNOPSIS

```
Diff(dstSet, srcSet: LargeSet);
```

dstSet - destination for difference operation

srcSet - source for difference operation

DESCRIPTION

Perform a set difference operation. The result of the difference of the two sets is stored back into the **dstSet**. Both sets must have exactly the same set range. This procedure corresponds directly to the standard Modula-2 '-' set operator.

EXAMPLES

```
VAR
```

```
  MySet, YourSet : LargeSet;
```

```
BEGIN
```

```
  IF (CreateSet(MySet, 0, 999)) AND (CreateSet(YourSet, 0, 999)) THEN
```

```
    ...
    Diff(MySet, YourSet);
```

```
    ...
    DeleteSet(MySet);
```

```
    DeleteSet(YourSet);
```

```
  END;
```

```
  ...
```

Modula-2 Software Construction Set Reference Guide

NAME

Intersection - perform an intersection operation on two large sets

SYNOPSIS

```
Intersection(dstSet, srcSet: LargeSet);
```

dstSet - destination for intersection operation

srcSet - source for intersection operation

DESCRIPTION

Perform a set intersection operation. The result of the intersection of the two sets is stored back into the **dstSet**. Both sets must have exactly the same set range. This procedure corresponds directly to the standard Modula-2 '*' set operator.

EXAMPLES

VAR

```
MySet, YourSet : LargeSet;
```

BEGIN

```
IF (CreateSet(MySet, 0, 999)) AND (CreateSet(YourSet, 0, 999)) THEN
```

```
    ...  
    Intersection(MySet, YourSet);
```

```
    ...  
    DeleteSet(MySet);
```

```
    DeleteSet(YourSet);
```

```
END;
```

...

NAME

SymmetricDiff - perform a symmetric difference operation on two large sets

SYNOPSIS

SymmetricDiff(dstSet, srcSet: LargeSet);

dstSet - destination for symmetric difference operation

srcSet - source for symmetric difference operation

DESCRIPTION

Perform a set symmetric difference operation. The result of the symmetric difference of the two sets is stored back into the **dstSet**. Both sets must have exactly the same set range. This procedure corresponds directly to the standard Modula-2 **'/'** set operator.

EXAMPLES

VAR

MySet, YourSet : LargeSet;

BEGIN

IF (CreateSet(MySet, 0, 999)) AND (CreateSet(YourSet, 0, 999)) THEN

...

SymmetricDiff(MySet, YourSet);

...

DeleteSet(MySet);

DeleteSet(YourSet);

END;

...

Chapter 21 Module LongInOut.....	3
ReadLongInt.....	4
ReadLongCard.....	5
WriteLongInt.....	6
WriteLongCard.....	6
WriteLongOct.....	6
WriteLongHex.....	6

Chapter 21 Module LongInt	3
ReadLongInt	4
ReadLongCard	5
WriteLongInt	6
WriteLongCard	6
WriteLongOct	6
WriteLongHex	6

The module **LongInOut** contains some logical extensions of the standard module **InOut**. The module consists of procedure for reading and writing values of type **LONGINT** and **LONGCARD** which are 32-bit versions of the standard 16-bit types **INTEGER** and **CARDINAL**. This module performs I/O operations by using the procedures in module **InOut** implicitly.

This module defines a variable **Done** which is set to **TRUE** or **FALSE** depending on if the previous call to a function in the module **LongInOut** was successful or unsuccessful.

For additional information about how the I/O is performed refer to the documentation for the module **InOut** which describes this in more detail.

```

EXAMPLES
VAR
  x : LONGINT;
BEGIN
  ...
  WriteString("Enter the x coordinate? ");
  ReadLongInt(x);
  IF NOT Done THEN
    WriteString("Invalid x coordinate!");
  END;
  ...

```

Modula-2 Software Construction Set Reference Guide

NAME

ReadLongInt - read a long integer and convert to binary

SYNOPSIS

ReadLongInt(VAR x: LONGINT);

x - long integer variable into which to store the binary value

DESCRIPTION

Read a string from the input stream. The string should contain an ASCII long integer. Convert the long integer to binary and store into the parameter x. The variable Done is set to TRUE if the long integer is read or FALSE if the operation failed.

The syntax for a long integer is:

longint = ["+"|"-"] digit {digit}.

EXAMPLES

```
VAR
  x : LONGINT;
BEGIN
  ...
  WriteString("Enter the x coordinate? ");
  ReadLongInt(x);
  IF NOT Done THEN
    WriteString("Invalid x coordinate!");
  END;
  ...
```

NAME

ReadLongCard - read a long cardinal and convert to binary

SYNOPSIS

ReadLongCard(VAR x: LONGCARD);

x - long cardinal variable into which to store the binary value

DESCRIPTION

Read a string from the input stream. The string should contain an ASCII long cardinal. Convert the long cardinal to binary and store into the parameter x. The variable Done is set to TRUE if the long cardinal is read or FALSE if the operation failed.

The syntax for a long cardinal is:

longcard = digit {digit}.

EXAMPLES

VAR

age : LONGCARD;

BEGIN

...

WriteString("Enter your age? ");

ReadLongCard(age);

IF NOT Done THEN

WriteString("Invalid Age!");

END;

...

Modula-2 Software Construction Set Reference Guide

NAME

WriteLongInt, WriteLongCard, WriteLongOct, WriteLongHex - output a number

SYNOPSIS

```
WriteLongInt(x: LONGINT; n: CARDINAL);  output x in decimal
WriteLongCard(x: LONGCARD; n: CARDINAL); output x in decimal
WriteLongOct(x: LONGCARD; n: CARDINAL);  output x in octal
WriteLongHex(x: LONGCARD; n: CARDINAL);  output x in hexadecimal
```

x - long integer or long cardinal binary number to be output
n - minimum number of characters to output

DESCRIPTION

Output an ASCII representation of a long integer or long cardinal number. The output is in decimal, octal, or hexadecimal. The parameter n specifies the minimum width of the output string. If the number of digits is less than the minimum specified then the field is padded with space characters. The **WriteLongHex** procedure always outputs 8 digits, this is done to make hexadecimal numbers easily distinguishable. The output is sent to the output stream.

EXAMPLES

```
VAR
  x: LONGCARD;
BEGIN
  ...
  WriteLongHex(x, 10);
  WriteLn;
  WriteLongCard(x, 5);
  ...
```

Chapter 22	Module MathLib0	3
arccos		4
arcsin		4
arctan		4
cos		4
sin		4
tan		4
cosh		5
sinh		5
tanh		5
exp		6
ln		6
log		6
power		6
sqrt		6
real		7
entier		7
DegToRad		8
RadToDeg		8

Chapter 22 Module MathLib	2
acos	4
asin	4
atan	4
cos	4
sin	4
tan	4
cosh	5
sinh	5
tanh	5
exp	6
ln	6
log	6
power	6
sqrt	6
real	7
entier	7
DeftMod	8
RedMod	8

The **MathLib0** module is a superset of the standard module available in most implementations. The module contains all of the standard procedures along with additional procedures and mathematical constants.

The procedures in this module take advantage of the Amiga "mathtrans.library". Therefore before any procedures in this module can be called the "mathtrans.library" must be opened successfully and the library base must be stored into the variable **MathTransBase**. Before exiting the Amiga resident library "mathtrans.library" must be explicitly closed. The easiest way to accomplish the above steps is to use the procedures in the module **InitMathLib0** described in another chapter of this manual. The following example demonstrates how to use **MathLib0**:

```
MODULE Demo;

FROM InitMathLib0 IMPORT OpenMathLib0, CloseMathLib0;
FROM MathLib0 IMPORT sin;
FROM Terminal IMPORT WriteString;

VAR
  x, y: REAL;
BEGIN
  IF OpenMathLib0() THEN
    ...
    x := sin(y);
    ...
    CloseMathLib0();
  ELSE
    WriteString("ERROR: 'mathtrans.library' not found!");
  END;
END Demo.
```

Instead of using the **InitMathLib0** module the following code may also be used to open and close the "mathtrans.library".

```
BEGIN
  ...
  MathTransBase := OpenLibrary(ADR(MathTransName), 00);
  IF (MathTransBase # NIL) THEN
    ...
    CloseLibrary(MathTransBase^);
  ELSE
    WriteString("ERROR: 'mathtrans.library' not found!");
  END;
  ...
```

Modula-2 Software Construction Set Reference Guide

NAME

arccos, arcsin, arctan, cos, sin, tan - trigonometric functions

SYNOPSIS

```
arccos(x: REAL): REAL;    return arccosine of x
arcsin(x: REAL): REAL;    return arcsine of x
arctan(x: REAL): REAL;    return arctangent of x
cos(x: REAL): REAL;       return cosine of x
sin(x: REAL): REAL;       return sine of x
tan(x: REAL): REAL;       return tangent x
```

x - angle in radians

result - value of trigonometric function

DESCRIPTION

These functions return the value of the corresponding trigonometric function for a given angle. The input value to these functions is not an angle in degrees but rather in radians.

EXAMPLES

```
VAR
  v, x, y: REAL;
BEGIN
  IF OpenMathLib0() THEN
    ...
    x := sin(v);
    y := tan(v);
    ...
    CloseMathLib0();
  END;
  ...
```

NAME

cosh, sinh, tanh - hyperbolic functions

SYNOPSIS

```

cosh(x: REAL): REAL;      return hyperbolic cosine of x
sinh(x: REAL): REAL;      return hyperbolic sine of x
tanh(x: REAL): REAL;      return hyperbolic tangent of x

```

x - angle in radians

result - value of hyperbolic function

DESCRIPTION

These functions return the value of the corresponding hyperbolic function for a given angle. The input value to these functions is not an angle in degrees but rather in radians.

EXAMPLES

```

VAR
  v, x, y: REAL;
BEGIN
  IF OpenMathLib0() THEN
    ...
    x := sinh(v);
    y := tanh(v);
    ...
    CloseMathLib0();
  END;
  ...

```

Modula-2 Software Construction Set Reference Guide

NAME

exp, ln, log, power, sqrt - logarithmic and exponential functions

SYNOPSIS

```
exp(x: REAL): REAL;    return exponential function of x
ln(x: REAL): REAL;     return natural log of x
log(x: REAL): REAL;     return naperian logarithm of x
power(x, y: REAL): REAL; return x to power of y
sqrt(x: REAL): REAL;    return square root of x
```

x - a real value

y - a real value

result - value of logarithmic or exponential function

DESCRIPTION

These functions return the value of the corresponding logarithmic or exponential function for given value(s).

EXAMPLES

VAR

v, x, y: REAL;

BEGIN

IF OpenMathLib0() THEN

...

x := sqrt(v);

y := exp(v);

...

CloseMathLib0();

END;

...

NAME

real, entier - type conversion functions

SYNOPSIS

real(x: LONGINT): REAL; convert a LONGINT to a REAL
 entier(x: REAL): LONGINT; convert a REAL to a LONGINT

x - value to be converted

result - value converted to a different type

DESCRIPTION

These functions convert a value between the types REAL and LONGINT. Conversion from REAL to LONGINT is performed by returning the integral part of the floating point number. The result of the **entier** function is unspecified when the REAL value is outside the range of LONGINT.

EXAMPLES

```
VAR
  x: REAL;
  l: LONGINT;
BEGIN
  IF OpenMathLib0() THEN
    ...
    x := real(l);
    l := entier(x);
    ...
    CloseMathLib0();
  END;
  ...
```

Modula-2 Software Construction Set Reference Guide

NAME

DegToRad, RadToDeg - radians and degrees conversion functions

SYNOPSIS

DegToRad(x: REAL): REAL; convert degrees to radians
RadToDeg(x: REAL): REAL; convert radians to degrees

x - value to be converted

result - value converted to the specified form

DESCRIPTION

These function convert a value from degrees to radians or from radians to degrees.

EXAMPLES

```
VAR
  v, x, y: REAL;
BEGIN
  IF OpenMathLib0() THEN
    ...
    x := DegToRad(v);
    y := RadToDeg(v);
    ...
    CloseMathLib0();
  END;
  ...
```


Chapter 23	Module RandomNumbers.....	3
	Random.....	4

Chapter 23 Modula-2 Random Number Generators

The **RandomNumbers** module implements a pseudo-random number generator. The pseudo-random number generator uses the linear congruential method to generate the random numbers. This method produces very good results.

The seed of the **RandomNumbers** module is automatically set to an arbitrary seed value. The seed can be changed at any time by storing a new value into the variable **Seed**. For a given seed value the same sequence of random numbers will be generated. To be sure of receiving a different random number sequence every time a program is run, load the seed value from the current system time value.

DESCRIPTION
Return a pseudo-random number between 0 and max-1. The algorithm for generating the random number is based on the linear congruential method.

EXAMPLES
The following example stores a random number between 0 and 999 into the variable x.

```
VAR
  x : LONGCARD;
BEGIN
  x := Random(1000);
```

Modula-2 Software Construction Set Reference Guide

NAME

Random - returns a random number within a specified range

SYNOPSIS

Random(max: LONGCARD): LONGCARD;

max - specifies the upper bounds of random number range

result - a random number in the specified range

DESCRIPTION

Return a pseudo-random number between 0 and max-1. The algorithm for generating the random number is based on the linear congruential method.

EXAMPLES

The following example stores a random number between 0 and 999 into the variable x.

VAR

x : LONGCARD;

BEGIN

x := Random(10000);

...

Chapter 24	Module RealConversions.....	3
	ConvStringToReal.....	4
	ConvRealToString.....	5

Chapter 24 Module RealConversions.....	2
ConvertingToReal.....	2
ConvertingToInteger.....	2

The **RealConversions** module provides functions for converting REAL to ASCII and ASCII to REAL. The functions in this module operate on strings and are not bound to any specific I/O device.

The REAL to ASCII conversion function defines a new type for specifying the output format:

TYPE

```
RealToStringFormat = (Scientific, Decimal, ShortestForm);
```

The **Scientific** form produces the following output:

```
[ - ]m.nE[+|-]xx
```

The **Decimal** form produces the following output:

```
[ - ]m.n
```

The **ShortestForm** uses either the **Scientific** or the **Decimal** form depending on which one can be represented in fewer characters.

NAME

ConvStringToReal - ASCII to floating point conversion

SYNOPSIS

ConvStringToReal(s: ARRAY OF CHAR): REAL;

s - string containing the ASCII floating point number

result - floating point number in binary form

DESCRIPTION

Convert ASCII floating point number to a binary floating point number. Leading blanks are ignored. The floating point string may contain a decimal point and may be followed by an "e" or "E" and a signed integer exponent.

EXAMPLES

The following example prompts the user for a string and then converts the string to a floating point value.

```
VAR input : ARRAY [0..99] OF CHAR; v : REAL;
```

```
BEGIN
```

```
  WriteString("Size ? ");
```

```
  ReadString(input);
```

```
  v := ConvStringToReal(input);
```

```
  ...
```

NAME

ConvRealToString - floating point to ASCII conversion

SYNOPSIS

```
ConvRealToString(x: REAL;  
                 VAR s: ARRAY OF CHAR;  
                 digits: CARDINAL;  
                 format: RealToStringFormat);
```

x - floating point number to be converted
s - string which is to receive the output
digits - number of digits to be output past the decimal point
format - the desired output format

DESCRIPTION

Convert a binary floating point number to an ASCII string. The string which is used for output must be large enough to accommodate the result. The output is performed according to the selected format. The parameter digits determines how many digits of precision to generate.

EXAMPLES

The following example demonstrates how a REAL number can be output to the terminal.

```
VAR outStr : ARRAY [0..79] OF CHAR; val : REAL;
```

```
BEGIN
```

```
...  
  ConvRealToString(val, outStr, 7, Decimal);  
  WriteString(outStr);  
...
```


Chapter 25	Module RealInOut.....	3
	ReadReal.....	4
	WriteReal.....	5

Chapter 25: Module Realization.....	25-2
Readers.....	25-2
Writers.....	25-2

The standard module **RealInOut** is available in most implementations. This module provides procedures for input and output of real numbers. The actual input and output is performed through the module **InOut**. For more information refer to the chapter on the module **InOut**.

This module defines a variable **Done** which is set to TRUE or FALSE depending on if the previous call to a function in the module **RealInOut** was successful or unsuccessful.

EXAMPLES

```
VAR  
Y : REAL;  
BEGIN  
  ReadReal(Y);  
  ...  
END
```

Modula-2 Software Construction Set Reference Guide

NAME

ReadReal - read a real and convert to binary representation

SYNOPSIS

```
ReadReal(VAR x: REAL);
```

x - variable into which the REAL number is to be stored

DESCRIPTION

Inputs a string representing a floating point number. The input is taken from the input stream established in the module `InOut`. A string to real conversion is performed and the result stored into the parameter x.

EXAMPLES

```
VAR
  y : REAL;
BEGIN
  ...
  ReadReal(y);
  ...
```


The standard module **RealInOut** is available in most implementations. This module provides procedures for input and output of real numbers. The actual input and output is performed through the module **InOut**. For more information refer to the chapter on the module **InOut**.

This module defines a variable **Done** which is set to **TRUE** or **FALSE** depending on if the previous call to a function in the module **RealInOut** was successful or unsuccessful.

EXAMPLES

```
VAR  
Y : REAL;  
BEGIN
```

```
ReadReal(Y);
```

NAME

WriteReal - output a real number

SYNOPSIS

WriteReal(x: REAL; n: CARDINAL);

x - REAL number to be output

n - minimum number of characters to output

DESCRIPTION

Converts a REAL number to a string. Send the string to the output stream, established in the module InOut. The output string will contain a specified minimum number of characters. The string will be padded with spaces if not enough digits are generated.

EXAMPLES

VAR

x: REAL;

BEGIN

...

WriteReal(x, 1);

...

Chapter 26	Module RunTimeErrors.....	3
	InstallErrorHandler.....	4
	RemoveErrorHandler.....	5

Chapter 26 Module Runtime Errors 26-2
InstallErrorHandler 26-2
RemoveErrorHandler 26-2

The purpose of this module is to provide support for run time error checking. Run time error checking is enabled by taking over a tasks `tcTrapCode` and `tcTrapData` fields of the `Task` structure. When a program encounters a trap or exception such as `CHK` or `TRAPV` a window is opened on the `WorkBench` screen, the window contains a register dump and other process related information. The user is given three options:

- "C" - continue execution from this point
- "E" - exit program immediately
- "P" - propagate the error to the next trap handler

For more information on run-time errors refer to the compiler chapter of this manual.

Module-2 Software Construction Set Reference Guide

NAME

InstallErrorHandler - install error handler

SYNOPSIS

InstallErrorHandler();

DESCRIPTION

Install an error handler on the current process. The error handler must be removed via a call to **RemoveErrorHandler** before exiting the program.

EXAMPLES

(* \$V+ enable overflow checking *)

VAR x, y : INTEGER;

BEGIN

InstallErrorHandler();

x := 30000;

y := 30000;

x := x + y; (* <- Run-Time Error: Overflow *)

RemoveErrorHandler();

...

NAME

RemoveErrorHandler - remove error handler

SYNOPSIS

RemoveErrorHandler();

DESCRIPTION

Remove the error handler from the current process. This operation must be performed before terminating a program. If the error handler was not installed no action is taken.

EXAMPLES

```
(* $V+ enable overflow checking *)  
VAR x, y : INTEGER;  
  
BEGIN  
  InstallErrorHandler();  
  x := 30000;  
  y := 30000;  
  x := x + y; (* <- Run-Time Error: Overflow *)  
  RemoveErrorHandler();  
  ...
```


Chapter 27 Module Storage.....	3
ALLOCATE.....	4
DEALLOCATE.....	5
Available.....	6

Chapter 27 Module Storage	2
ALLOCATE	4
DEALLOCATE	5
Available	6

The standard Modula-2 module **Storage** has been implemented to allow programs from other environments to be moved to the Amiga. Most implementations of Modula-2 implement the module **Storage**.

This module allocates memory using the ROM Kernel **AllocMem** function with the memory type **MemReqSet{}**. This memory type tries to allocate fast memory first if available, otherwise chip memory is allocated.

Some programs written on other systems allocate memory dynamically but do not deallocate the memory. These programs assume that the operating system will deallocate all the memory automatically when the program exits. This is not the case with the Amiga operating system, to allow such programs to work properly on the Amiga the **Heap** module should be used. The **Heap** module keeps track of all memory allocations and allows all of the memory currently allocated to be released with one procedure call. For more information refer to the chapter of this manual which describes the **Heap** module.

Modula-2 Software Construction Set Reference Guide

NAME

ALLOCATE - allocate an area of a given size

SYNOPSIS

```
ALLOCATE(VAR a: ADDRESS; size: LONGCARD);
```

a - store address of memory allocated or NIL if no memory

size - number of bytes to allocate

DESCRIPTION

Allocate a block of memory of a specified **size**. The memory is allocated by calling the ROM Kernel **AllocMem** function with a **MemReqSet{}**. If the memory could not be allocated a NIL is returned as the address.

EXAMPLES

```
VAR Table : POINTER TO ARRAY [0..1023] OF BOOLEAN;
```

```
BEGIN
```

```
  ALLOCATE(Table, 10240);
```

```
  IF Table = NIL THEN
```

```
    WriteString("insufficient memory to continue!");
```

```
  END;
```

```
  ...
```

NAME

DEALLOCATE - return memory allocated by ALLOCATE

SYNOPSIS

DEALLOCATE(VAR a: ADDRESS; size: LONGCARD);

a - address of the memory to be deallocated

size - number of bytes to free

DESCRIPTION

Deallocates memory allocated with **ALLOCATE**. The memory block being freed must be exactly the same as the block returned by **ALLOCATE**.

EXAMPLES

VAR Table : POINTER TO ARRAY [0..1023] OF BOOLEAN;

BEGIN

ALLOCATE(Table, 1024D);

IF Table = NIL THEN

WriteString("insufficient memory to continue!");

END;

...

DEALLOCATE(Table, 1024D);

...

Modula-2 Software Construction Set Reference Guide

NAME

Available - determine if a specified amount of memory is free

SYNOPSIS

```
Available(size: LONGCARD): BOOLEAN;
```

size - number of bytes that needs to be available

result - TRUE if the specified number of bytes are available

DESCRIPTION

Check if a specified number of bytes are available as contiguous memory. If the memory has become fragmented then this procedure may return FALSE even when the total amount of free memory exceeds the specified size. Modula-2 programs moved from other system may use this function before using the **ALLOCATE** function.

EXAMPLES

```
VAR Table : ARRAY [0..1999] OF CHAR;
```

```
BEGIN
```

```
  ...  
  IF Available(20000) THEN  
    ALLOCATE(Table, 20000);  
  END;  
  ...
```


Chapter 28	Module Strings	3
	StringLength	4
	ConvStringToUpperCase	5
	CompareString	6
	CompareStringCAP	7
	CopyString	8
	ConcatString	9
	InsertSubString	10
	OverwriteWithSubString	11
	DeleteSubString	12
	ExtractSubString	13
	LocateSubString	14
	LocateChar	15

Chapter 28 Module Strings.....3

StringLength.....4

ConvertStringToUpperCase.....5

CompareString.....6

CompareStringCase.....7

CopyString.....8

ConcatString.....9

InsertSubString.....10

OverwriteSubString.....11

DeleteSubString.....12

ExtractSubString.....13

LocateSubString.....14

LocateChar.....15

The **Strings** module provides many functions for manipulating strings. A string is defined as a group of characters followed by a null character. Some of the procedures in the **Strings** module do not check if a string has become larger than the amount of memory actually allocated for the string. Therefore it becomes very important to reserve enough space so that a string will never become too large and overflow its buffer.

The **Strings** module has several procedures used in comparing one string to another. These procedures return a result which indicates if the strings are equal, less, or greater. The result is defined as follows:

TYPE

Relation = (less, equal, greater);

The strings are considered **equal** if they are of the same length and the characters representing the strings are identical. If the strings are not identical then one string will be considered **less** or **greater** than the other string. The following example will illustrate the possible result of a string compare function:

```
"abc" = "abc"
"ABC" < "abc"
"abc" > "ABC"
```

Modula-2 Software Construction Set Reference Guide

NAME

StringLength - determine the length of a string

SYNOPSIS

StringLength(string: ARRAY OF CHAR): CARDINAL;

string - string to be measured

result - length of string

DESCRIPTION

Count the number of characters in a string. This is done by starting from the first character of the string and continuing until a null character is reached. A string which consists of only the null character has a length of zero.

EXAMPLES

```
VAR s : ARRAY [0..127] OF CHAR; len : CARDINAL;
```

```
BEGIN
```

```
  ReadString(s);
```

```
  len := StringLength(s);
```

```
  ...
```

NAME

ConvStringToUpperCase - convert a string to upper case

SYNOPSIS

ConvStringToUpperCase(VAR string: ARRAY OF CHAR);

string - string to be converted to upper case

DESCRIPTION

This procedure performs a Modula-2 CAP() standard function on each character in the string, until a null character is encountered.

EXAMPLES

An example of using **ConvStringToUpperCase** is when a program requests the user to type the name of a command. The user might use upper or lower case characters to input the command. Your program can then convert the input string to all upper case, then to determine which command the user has entered it is only necessary to compare the input string to the upper case strings representing valid commands.

VAR in : ARRAY [0..79] OF CHAR;

BEGIN

 ReadString(in);

 ConvStringToUpperCase(in);

 IF CompareString(in, "LIST") = equal THEN

 ...
 ELSIF CompareString(in, "RUN") = equal THEN

 ...
 END;

 ...

Module-2 Software Construction Set Reference Guide

NAME

CompareString - compare two strings (case sensitive)

SYNOPSIS

CompareString(string1, string2: ARRAY OF CHAR): Relation;

string1 - the string on the left side of the compare

string2 - the string on the right side of the compare

result - result of comparing the strings is a Relation

DESCRIPTION

Compare two strings character by character. The strings are equal if they are of exactly the same length and the characters in the strings are identical.

EXAMPLES

```
VAR cmd : ARRAY [0..79] OF CHAR;
```

```
BEGIN
```

```
  ReadString(cmd);
```

```
  IF CompareString(cmd, "EXIT") = equal THEN
```

```
    WriteString("Done...");
```

```
    HALT;
```

```
  END;
```

```
...
```

NAME

CompareStringCAP - compare two strings (case insensitive)

SYNOPSIS

CompareStringCAP(string1, string2: ARRAY OF CHAR): Relation;

string1 - the string on the left side of the compare

string2 - the string on the right side of the compare

result - result of comparing the strings is a Relation

DESCRIPTION

Compare two strings to each other regardless of case. The two strings are equal if they are of the same length and if the characters in the strings are identical, regardless of lower or upper case.

EXAMPLES

In the following example the `CompareStringCAP` procedure will return `equal` because the strings only differ in the case of the letters.

```
VAR flag : BOOLEAN;
```

```
BEGIN
```

```
...  
  flag := CompareStringCAP("Abc","AbC") = equal;  
...
```

Modula-2 Software Construction Set Reference Guide

NAME

CopyString - copy a string from one buffer to another buffer

SYNOPSIS

CopyString(VAR toString: ARRAY OF CHAR; fromString: ARRAY OF CHAR);

toString - the destination string for the copy

fromString - the source string for the copy

DESCRIPTION

Copy the characters from the **fromString** to the **toString**. The **fromString** is truncated if it is larger than the **toString**.

EXAMPLES

VAR name, input : ARRAY [0..79] OF CHAR;

BEGIN

```
...  
  CopyString(name, input);  
...
```


NAME

ConcatString - append a string to the end of another string

SYNOPSIS

ConcatString(VAR toString: ARRAY OF CHAR; fromString: ARRAY OF CHAR);

toString - string to which the fromString is to be appended to
 fromString - string which is to be appended to the toString

DESCRIPTION

The characters of the fromString are copied to the toString beginning after the last character of the toString. If the toString is not large enough to accommodate the new string the toString will be truncated.

EXAMPLES

VAR fileName : ARRAY [0..59] OF CHAR;

BEGIN

... ConcatString(fileName, ".DOC");

...

Modula-2 Software Construction Set Reference Guide

NAME

InsertSubString - insert one string into another string

SYNOPSIS

```
InsertSubString(VAR baseString: ARRAY OF CHAR;  
                subString: ARRAY OF CHAR; start: CARDINAL);
```

baseString - string into which to insert

subString - string to be inserted

start - the baseString char after which to begin insert

DESCRIPTION

Insert a **subString** into a **baseString** at a specified position. The characters after the **start** position are shifted over to make room for the **subString**. If the **baseString** is not large enough to accommodate the new **subString** those characters which do not fit into the **baseString** are truncated. If **start** is greater than **StringLength(baseString)** then the result is undefined.

EXAMPLES

In this example if the string **inputFileName** contained "test.mod" after the **InsertSubString** procedure call the **inputFileName** contains "RAM:test.mod".

```
VAR inputFileName : ARRAY [0..79] OF CHAR;
```

```
BEGIN
```

```
...  
  InsertSubString(inputFileName, "RAM:", 0);
```

```
...
```

NAME

OverwriteWithSubString - overwrite a string with another string

SYNOPSIS

```
OverwriteWithSubString(VAR baseString: ARRAY OF CHAR;
                       subString: ARRAY OF CHAR;
                       start: CARDINAL);
```

baseString - string to be partially overwritten

subString - string to overwrite with

start - character at which to begin overwriting baseString

DESCRIPTION

The **baseString** is overwritten by the **subString** beginning with the character at position **start**. The length of the **baseString** may increase after being overwritten. The **subString** will be truncated if it goes beyond the bounds of the array containing the **baseString**.

EXAMPLES

The **fileName** variable contains "df0:test.doc" after the **OverwriteWithSubString** is performed the **fileName** variable contains "df3:test.doc".

```
VAR fileName : ARRAY [0..79] OF CHAR;
```

```
BEGIN
```

```
...
  OverwriteWithSubString(fileName, "df3:", 0);
...

```

Modula-2 Software Construction Set Reference Guide

NAME

DeleteSubString - delete characters from a string

SYNOPSIS

```
DeleteSubString(VAR baseString: ARRAY OF CHAR;  
                fromPosition: CARDINAL; length: CARDINAL);
```

baseString - string from which to delete chars

fromPosition - character position from which to begin delete

length - number of chars to delete

DESCRIPTION

Delete a specified number of characters from the middle of a string. Those characters which come after the deleted characters are shifted down to fill the space previously occupied by the deleted characters.

EXAMPLES

The variable **patStr** contains "abcdef" after the **DeleteSubString** procedure is applied, the **patStr** becomes "abf".

```
VAR patStr : ARRAY [0..39] OF CHAR;
```

BEGIN

```
...  
DeleteSubString(patStr, 2, 3);
```

```
...
```

NAME

ExtractSubString - extract characters from a string

SYNOPSIS

```
ExtractSubString(VAR subString: ARRAY OF CHAR;
                 baseString: ARRAY OF CHAR;
                 fromPosition: CARDINAL;
                 length: CARDINAL);
```

subString - store extracted characters into this string

baseString - string from which to extract the substring

fromPosition - begin extracting from a specified position

length - number of characters to extract

DESCRIPTION

This procedure copies a specified number of characters from a **baseString** to a **subString** starting at a specified position. If a null character is encountered in the **baseString** before the specified number of characters are extracted then the **subString** is terminated at that point.

EXAMPLES

If **cmdStr** contains "LOAD:test.dat" then executing an **ExtractSubString** procedure will result in an extracted substring of "test.dat".

```
VAR extStr, cmdStr : ARRAY [0..39] OF CHAR;
```

```
BEGIN
```

```
...
  ExtractSubString(cmdStr, subStr, 5, 8);
...

```

Modula-2 Software Construction Set Reference Guide

NAME

LocateSubString - search for a substring

SYNOPSIS

```
LocateSubString(baseString, subString: ARRAY OF CHAR;  
                start, end: CARDINAL): INTEGER;
```

baseString - string to be used for the search

subString - substring to search for

start - starting position for search

end - ending position for search

result - position of **subString** in **baseString** or -1 if not found

DESCRIPTION

Search for the occurrence of a **subString** inside a **baseString**. The search begins at the start position and ends at the specified end position. The result of the search is either the position of the first occurrence of the **subString** in the **baseString** or -1 if the **subString** was not found in the **baseString**. If the **subString** is smaller than the **baseString** the result is -1.

EXAMPLES

```
VAR input : ARRAY [0..79] OF CHAR; flag : BOOLEAN;
```

```
BEGIN
```

```
  ...  
  flag := LocateSubString(input, "***", 0, 79) # -1;
```

```
  ...
```

NAME

LocateChar - search for a character in a string

SYNOPSIS

```
LocateChar(string: ARRAY OF CHAR; c: CHAR;  
           start, end: CARDINAL): INTEGER;
```

string - string to be searched for character

c - character to search for

start - starting position of search in string

end - ending position of search in string

result - position of char, or -1 if char not found in string

DESCRIPTION

Search a string from a specified start to a specified end position for the first occurrence of a character. If the character is found in the string then the position is returned, otherwise -1 is returned to indicate that the character was not found.

EXAMPLES

```
VAR input : ARRAY [0..79] OF CHAR; flag : BOOLEAN;
```

```
BEGIN
```

```
...  
  flag := LocateChar(input, ".", 0, 79) # -1;  
...
```


Chapter 29 Module System.....3

The purpose of the **System** module is to provide basic run-time support functions. These functions include 32-bit arithmetic functions, floating point functions, and the startup code necessary to run under the AmigaDOS operating system.

The first code executed in a Modula-2 program is the startup code which resides in the **System** module. The startup begins by storing the contents of registers D0 and A0 into the variables **CmdLineLength** and **CmdLinePtr**. The two registers are initialized by AmigaDOS to contain the length of the CLI command line and the pointer to the CLI command line string, these two variables will not be valid if the program was invoked from the WorkBench environment. To allow a program to determine how much stack space it has been given the variables **StackPtr** and **StackSize** are initialized with the top of stack pointer and the size of the stack respectively. Next, the startup code opens the following Amiga libraries: "exec.library", "dos.library", "mathffp.library", "intuition.library" and "graphics.library", "layers.library". The next step is for the startup code to determine the environment in which the program is being executed, the two possible environments are CLI and WorkBench.

First, we will cover what happens when a program is started from the CLI environment. The startup code will parse the command line arguments. An argument is considered to be one or more characters separated by a blank space character, the exception to this rule are characters enclosed in double quotes which are considered as one argument. Each argument is copied into a buffer and followed by a null character, a pointer to the argument is stored into the **argv** array beginning at index 1. Index 0 of the **argv** array contains a pointer to the name of the program. The double quotes which may enclose an argument are not stripped away and are considered part of the argument string. After the arguments are parsed the variable **argc** is set to the number of arguments found on the command line, including the program name itself. Therefore a program invoked from the CLI environment will have an **argc** of greater or equal to one. The final action of the CLI startup code is to initialize the variables **StdInput** and **StdOutput** by storing the values returned by the AmigaDOS functions **Input** and **Output** respectively, these functions return the programs standard input and standard output file handles. The user of a program may redirect the standard input and output to another device or file by using the ">" and "<" symbols followed by the name of a file or device. For more information about redirection refer to the AmigaDOS manual.

If the program has been invoked by WorkBench then an entirely different set of functions must be performed. After invoking the process, WorkBench sends a startup message to our process. This message is sent to our processes' message port. The startup code performs a **WaitPort** function on the message port and then a **GetMsg** function to retrieve the message. A pointer to the message is stored into the variable **WBenchMsg**. An AmigaDOS window is then opened and the variables **StdInput** and **StdOutput** are assigned the file handle returned by AmigaDOS for that window. The window is opened to allow a program which performs output to work properly.

To determine if your program has been invoked from the CLI or from WorkBench test the variable **argc**. If **argc** is equal to zero then you are running under WorkBench, otherwise **argc** contains the number of arguments from the CLI.

After the startup initialization has been finished the pointer to the **System** module's base data address is loaded into the register A4 and JSR is performed to the module initialization JSR table. At the end of this table is a JMP to the main program module.

When a program has finished executing or a HALT has been encountered execution returns to the **System** module. The **System** module is responsible for final cleanup before exiting to the operating system. The cleanup consists of closing the Amiga libraries initially opened by the startup code. If the program was started from WorkBench then the output window opened by the startup code is closed and the message initially sent by WorkBench is replied. Finally, the contents of variable **CLIReturnCode** is loaded into register D0 and the program exits to AmigaDOS.

The procedures defined in module **System** are used only for run-time support and should not be called directly by a program. These procedures are only documented in the definition module and need no further explanations. Do not call any of the procedures in this module directly.

If the program has been invoked by Workbench then an entirely different set of functions must be performed. After invoking the process, Workbench sends a startup message to our process. This message is sent to our process' message port. The startup code performs a WaitPort function on the message port and then a GetMsg function to retrieve the message. A pointer to the message is stored into the variable **WbStartup**. An AmigaDOS window is then opened and the variables **StdInout** and **StdOutput** are assigned the file handles returned by AmigaDOS for that window. The window is opened to allow a program which performs output to work properly.

To determine if your program has been invoked from the CLI or from Workbench test the variable **argc**. If **argc** is equal to zero then you are running under Workbench, otherwise **argc** contains the number of arguments from the CLI.

Chapter 30	Module Terminal.....	3
	Read.....	4
	BusyRead.....	5
	ReadAgain.....	6
	Write.....	7
	WriteLn.....	8
	WriteString.....	9

Chapter 35 Module Terminal 3
Read 4
BusyRead 5
ReadAgain 6
Write 7
WriteLn 8
WriteString 9

The standard module **Terminal** is available on most implementations of Modula-2. This module provides simple input and output procedures for reading and writing text.

The input and output is done through the **StdInput** and **StdOutput** file handles defined in the module **System**. By using the AmigaDOS redirection symbols ">" and "<" the user can redirect either the input or output to a file or another device.

The variable **eof** defined by this module can be checked to determine if the end of a file has been reached. This flag may be set after a **Read** operation. This flag is set to **TRUE** if the user has redirected the input to an AmigaDOS file and all the characters from the file have been read. It should also be mentioned that by checking this flag your program will probably not be directly portable to other systems, because this flag may not be implemented on other systems.

The **Read** and **BusyRead** functions will behave differently depending on if the program is performing input from a "RAW:" or "CON:" window. In a "RAW:" window characters will not be echoed to the screen and will be available immediately after being typed. In a "CON:" window characters will be echoed to the screen and will not be available until the user types <RETURN>.

Modula-2 Software Construction Set Reference Guide

NAME

Read - read a character from the terminal

SYNOPSIS

Read(VAR ch: CHAR);

ch - variable which is to receive the next character

DESCRIPTION

Request a character from the terminal. The program will wait until the user presses a key.

EXAMPLES

VAR ch : CHAR;

BEGIN

...
 Read(ch);
 ...

NAME

BusyRead - read character from terminal if ready

SYNOPSIS

BusyRead(VAR ch: CHAR);

ch - variable into which the character will be stored

DESCRIPTION

If a character is available return it immediately, if no character is available then return the value zero. This procedure should only be used when the **StdInput** is connected to an interactive terminal. The procedure will not work when **StdInput** is a file or a non interactive device.

EXAMPLES

The following example will continue to output the character "." until the user types a key at the terminal.

```
VAR cmd : CHAR;
```

```
BEGIN
```

```
...
```

```
  LOOP
```

```
    BusyRead(cmd);
```

```
    IF cmd # 0C THEN EXIT; END;
```

```
    Write(".");
```

```
  END;
```

```
...
```

Modula-2 Software Construction Set Reference Guide

NAME

ReadAgain - cause the last character Read to be Read again

SYNOPSIS

ReadAgain;

DESCRIPTION

This procedure pushes back the last character returned by **Read** or **BusyRead** back onto the input stream. The next call to **Read** or **BusyRead** will return the same character again.

EXAMPLES

In this example after the user types a character, the program will display the same character twice.

```
VAR cmd : CHAR;
```

```
BEGIN
```

```
...  
  Read(cmd);  
  Write(cmd);  
  ReadAgain;  
  Read(cmd);  
  Write(cmd);  
...
```

NAME

Write - output a character to the terminal

SYNOPSIS

Write(ch: CHAR);

ch - character to be output to the terminal

DESCRIPTION

Output a character to the terminal. The specified character is sent directly to AmigaDOS without any translations.

EXAMPLES

BEGIN

```
Write("1");  
Write("2");  
Write("3");  
...
```

Modula-2 Software Construction Set Reference Guide

NAME

WriteLn - terminate the current line of output

SYNOPSIS

```
WriteLn;
```

DESCRIPTION

Terminate the current output line. On the Amiga this is done by sending a line feed to the terminal.

EXAMPLES

The following example will cause "One..." to be printed on the terminal, and "Two..." to be printed on the next line of terminal.

BEGIN

```
WriteString("One...");  
WriteLn;  
WriteString("Two...");  
WriteLn;  
...
```

NAME

WriteString - write a string to the terminal

SYNOPSIS

WriteString(s: ARRAY OF CHAR);

s - string to be output to terminal

DESCRIPTION

Output the characters of a string to the terminal until a null character is encountered. The null character is not output. No line feed is generated after the string is printed.

EXAMPLES

The following example will output "Hello World!" to the terminal.

BEGIN

WriteString("Hello World!");

...

Chapter 31	Module TermInOut.....	3
	Read.....	4
	ReadString.....	5
	ReadInt.....	6
	ReadCard.....	7
	Write.....	8
	WriteLn.....	9
	WriteString.....	10
	WriteInt.....	11
	WriteCard.....	11
	WriteOct.....	11
	WriteHex.....	11

Chapter 31 Module Terminology 31
Read 31
ReadString 31
ReadInt 31
ReadCard 31
Write 31
WriteLn 31
WriteString 31
WriteInt 31
WriteCard 31
WriteOct 31
WriteHex 31

This module is very similar to the standard **InOut** module. The main difference is that it only support i/o to the terminal. The main advantage of this module is that it is a fraction of the size of the full **InOut** module, and it does not import any other modules. This module only adds 1.5K of code to your program a big savings over using **InOut** which also pulls in **FileSystem**, **Terminal**, **Memory**, **AmigaDOS**.

The input stream is connected to the terminal **StdInput** and the output stream is connected to the terminal **StdOutput**. This means that any procedure which performs a write will send data to the users terminal and any procedure which performs a read will get its input from the users terminal.

This module defines a variable **Done** which is set to **TRUE** or **FALSE** depending on if the previous call to a function in the module **TermInOut** was successful or unsuccessful. The variable **termCH** is set to the character which terminated input by the **ReadString**, **ReadInt** and **ReadCard** functions. The variable **Echo** controls the operation of the **ReadString** procedure. When **Echo** is set to **TRUE** **ReadString** will echo any input, when **FALSE** the input will not be echoed. By default the variable is set to **TRUE**. The value to be used for **Echo** depends on the type of output window through which the i/o is taking place. If the i/o is going through a "CON:" window, like the one used by the CLI the **Echo** variable should be set to **FALSE**. When performing i/o through a "RAW:" window, then the **Echo** variable should be set to **TRUE**.

The module **TermInOut** defines the constant **EOL**, this represents the code for the end of line character for this implementation of the module **TermInOut**.

Modula-2 Software Construction Set Reference Guide

NAME

Read - read a char from the input stream

SYNOPSIS

```
Read(VAR ch: CHAR);
```

ch - variable to store the next character in

DESCRIPTION

Read the next character from the input stream and store it into the parameter ch. Set the variable Done to TRUE if the operation was successful. If an end of file condition was detected set Done to FALSE.

EXAMPLES

```
VAR cmdCh : CHAR;
```

```
BEGIN
```

```
...  
LOOP
```

```
  Read(cmdCh);
```

```
  IF NOT Done THEN
```

```
    EXIT;
```

```
  END;
```

```
...  
END;
```

```
...
```

NAME

ReadString - read a string from the input stream

SYNOPSIS

ReadString(VAR s: ARRAY OF CHAR);

s - array which is to receive the input string

DESCRIPTION

Read a sequence of characters not containing blank or control characters. Leading blanks are ignored. Input is terminated by a carriage return or blank space. To delete a character the backspace key can be used. The character which caused the input to be terminated is stored in termCH.

EXAMPLES

The following example prompts the user for a command, then reads the command from the input stream.

VAR cmd : ARRAY [0..127] OF CHAR;

BEGIN

...
 WriteString("Enter next Command? ");
 ReadString(cmd);
...

Modula-2 Software Construction Set Reference Guide

NAME

ReadInt - read an integer and convert to binary

SYNOPSIS

ReadInt(VAR x: INTEGER);

x - integer variable into which to store the binary value

DESCRIPTION

Read a string from the input stream. The string should contain an ASCII integer. Convert the integer to binary and store into the parameter x. The variable Done is set to TRUE if the integer is read or FALSE if the operation failed.

The syntax for an integer is:

integer = ["+"|"-"] digit {digit}.

EXAMPLES

VAR x : INTEGER;

BEGIN

```
...
  WriteString("Enter the x coordinate? ");
  ReadInt(x);
  IF NOT Done THEN
    WriteString("Invalid x coordinate!");
  END;
...
```

NAME

ReadCard - read a cardinal and convert to binary

SYNOPSIS

ReadCard(VAR x: CARDINAL);

x - cardinal variable into which to store the binary value

DESCRIPTION

Read a string from the input stream. The string should contain an ASCII cardinal. Convert the cardinal to binary and store into the parameter x. The variable Done is set to TRUE if the cardinal is read or FALSE if the operation failed.

The syntax for a cardinal is:

cardinal = digit {digit}.

EXAMPLES

VAR age : CARDINAL;

BEGIN

```
...
  WriteString("Enter your age? ");
  ReadCard(age);
  IF NOT Done THEN
    WriteString("Invalid Age!");
  END;
...
```

Modula-2 Software Construction Set Reference Guide

NAME

Write - write a character to the output stream

SYNOPSIS

```
Write(ch: CHAR);
```

ch - the character to be sent to the output stream

DESCRIPTION

Output a character to the output stream.

EXAMPLES

The following example outputs the string "12" to the output stream.

BEGIN

```
...  
  Write("1");  
  Write("2");  
...
```

NAME

WriteLn - terminate the current line of the output stream

SYNOPSIS

WriteLn;

DESCRIPTION

Terminate the current output line by sending a line feed to the output stream.

EXAMPLES

The following example will cause "One..." to be sent to the output stream and "Two..." to be sent to the next line.

BEGIN

```
...
WriteString("One...");
WriteLn;
WriteString("Two...");
WriteLn;
...
```

Modula-2 Software Construction Set Reference Guide

NAME

WriteString - output a string to the output stream

SYNOPSIS

WriteString(s: ARRAY OF CHAR);

s - string to be output to the output stream

DESCRIPTION

Output a string to the output stream. The characters of the string are output until a null character is encountered. No line feed is generated after the string is output.

EXAMPLES

BEGIN

```
...  
  WriteString("Hello World!");  
...
```


NAME

WriteInt, WriteCard, WriteOct, WriteHex - write a number to output stream

SYNOPSIS

```
WriteInt(x,n: INTEGER);      write x in decimal to output stream
WriteCard(x,n: CARDINAL);    write x in decimal to output stream
WriteOct(x,n: CARDINAL);     write x in octal to output stream
WriteHex(x,n: CARDINAL);     write x in hexadecimal to output stream
```

x - integer or cardinal binary number to be output

n - minimum number of characters to output

DESCRIPTION

Output an ASCII representation of an integer or cardinal number. The output is in decimal, octal or hexadecimal. The parameter n specifies the minimum width of the output string. If the number of digits is less than the minimum specified then the field is padded with space characters. The WriteHex procedure always outputs 4 digits, this is done to make hexadecimal numbers easily distinguishable. The output is sent to the output stream.

EXAMPLES

```
VAR
  x: CARDINAL;
BEGIN
  ...
  WriteHex(x, 4);
  WriteLn;
  WriteCard(x, 5);
  ...
```


Lists.....	126
LongInOut.....	128
MathLib0.....	129
Memory.....	131
MiscResource.....	134
NarratorDevice.....	135
Nodes.....	137
ParallelDevice.....	138
Ports.....	139
PortsUtil.....	141
PotgoResource.....	142
Preferences.....	143
PrinterBase.....	147
PrinterDevice.....	149
RandomNumbers.....	152
Rasters.....	153
RealConversions.....	155
RealInOut.....	156
Regions.....	157
Resident.....	159
Resources.....	160
RomLayers.....	161
RunTimeErrors.....	162
Semaphores.....	163
SerialDevice.....	165
Sprites.....	167
Storage.....	169
Strings.....	170
System.....	172
Tasks.....	174
TasksUtil.....	177
Terminal.....	178
TermInOut.....	179
Text.....	180
TimerDevice.....	182
TrackDiskDevice.....	183
Translator.....	185
Views.....	186
Workbench.....	189

Chapter 32	Definition Modules	3
ADKHardware		4
Alerts		5
AmigaDOS		8
AmigaDOSExt		14
AmigaDOSProcess		16
Areas		19
ASCII		21
AudioDevice		22
Blit		23
BootBlock		27
CIAHardware		28
CIAResource		30
ClipboardDevice		31
Clipping		32
CList		34
ConfigRegs		37
ConsoleDevice		40
ConsoleDevUnit		42
Conversions		43
Copper		44
CopperUtil		46
CustomHardware		47
DiskFont		49
DiskResource		51
Display		52
DMAHardware		53
Drawing		54
Exec		57
ExecBase		58
Expansion		60
FileHandler		63
FileSystem		65
GamePortDevice		67
Gels		68
Graphics		74
GraphicsBase		75
Heap		77
InitMathLib0		78
InOut		79
InputDevice		81
InputEvents		82
Interrupts		84
INTHardware		86
Intuition		87
IntuitionBase		111
IODevices		113
IODevicesUtil		116
KeyboardDevice		117
KeyMapResource		118
LargeSets		119
Layers		121
Libraries		124

Module ADKHardware

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*****
* Name: ADKHardware.DEF      Version: Amiga.00.00 *
* Created: 11/29/86    Updated: 11/29/86    Author: Leon Frenkel *
* Description: Amiga ADK control definitions. *
*****)
```

DEFINITION MODULE ADKHardware;

CONST

```
ADKSetClr   = 15; (* standard set/clear bit *)
ADKPreComp1 = 14; (* two bits of precompensation *)
ADKPreComp0 = 13;
ADKMFMPPrec = 12; (* use mfm style precompensation *)
ADKUARTBrk  = 11; (* force uart output to zero *)
ADKWordSync = 10; (* enable DSKSYNC register matching *)
ADKMSBSync  = 9;  (* (Apple GCR Only) sync on MSB for reading *)
ADKFast     = 8;  (* 1 -> 2 us/bit (mfm), 2 -> 4 us/bit (gcr) *)
ADKUse3PN   = 7;  (* use aud chan 3 to modulate period of ?? *)
ADKUse2P3   = 6;  (* use aud chan 2 to modulate period of 3 *)
ADKUse1P2   = 5;  (* use aud chan 1 to modulate period of 2 *)
ADKUse0P1   = 4;  (* use aud chan 0 to modulate period of 1 *)
ADKUse3VN   = 3;  (* use aud chan 3 to modulate volume of ?? *)
ADKUse2V3   = 2;  (* use aud chan 2 to modulate volume of 3 *)
ADKUse1V2   = 1;  (* use aud chan 1 to modulate volume of 2 *)
ADKUse0V1   = 0;  (* use aud chan 0 to modulate volume of 1 *)

ADKPre000NS = {}; (* 000 ns of precomp *)
ADKPre140NS = {ADKPreComp0}; (* 140 ns of precomp *)
ADKPre280NS = {ADKPreComp1}; (* 280 ns of precomp *)
ADKPre560NS = {ADKPreComp0, ADKPreComp1}; (* 560 ns of precomp *)
```

END ADKHardware.

```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: Alerts.DEF               Version: Amiga.00.00                                     *
* Created: 11/17/86   Updated: 11/17/86   Author: Leon Frenkel                             *
* Description: Exec Alerts Types/Functions.                                                 *
*****)

```

DEFINITION MODULE Alerts;

(* Format of the alert error number:

```

* +-----+-----+-----+
* |D| SubSysId | General Error | SubSystem Specific Error |
* +-----+-----+-----+

```

```

*           D: DeadEnd alert
*           SubSysId: indicates ROM subsystem number
*           General Error: roughly indicates what the error was
*           Specific Error: indicates more detail
*)

```

CONST

```

(*)
(* General Dead-End Alerts *)
(*)

```

(* alert types *)

```

ATDeadEnd   = 80000000H;
ATRecovery  = 00000000H;

```

(* general purpose alert codes *)

```

AGNoMemory  = 00010000H;
AGMakeLib   = 00020000H;
AGOpenLib   = 00030000H;
AGOpenDev   = 00040000H;
AGOpenRes   = 00050000H;
AGIOError   = 00060000H;
AGNoSignal  = 00070000H;

```

(* alert objects *)

```

AExecLib    = 00008001H;
AGraphicsLib = 00008002H;
ALayersLib  = 00008003H;
AIntuition  = 00008004H;
AMathLib    = 00008005H;
AOLListLib  = 00008006H;
AODSLib     = 00008007H;
AORAMLib    = 00008008H;
AIconLib    = 00008009H;
AExpansionLib = 0000800AH;
AAudioDev   = 00008010H;
AConsoleDev = 00008011H;
AGamePortDev = 00008012H;
AKeyboardDev = 00008013H;
ATrackDiskDev = 00008014H;
ATimerDev   = 00008015H;
ACIARsrc    = 00008020H;
ADiskRsrc   = 00008021H;
AMiscRsrc   = 00008022H;
ABootStrap  = 00008030H;
AWorkbench  = 00008031H;

```


Module Alerts

```
(*
(* Specific Dead-End Alerts *)
*)

(* exec.library *)
ANExecLib      = 81000000H;
ANExcpVect     = 81000001H; (* 68000 exception vector checksum *)
ANBaseChkSum   = 81000002H; (* execbase checksum *)
ANLibChkSum    = 81000003H; (* library checksum failure *)
ANLibMem       = 81000004H; (* no memory to make library *)
ANMemCorrupt   = 81000005H; (* corrupted memory list *)
ANIntrMem      = 81000006H; (* no memory for interrupt servers *)
ANInitAPtr     = 81000007H; (* InitStruct() of an APTR source *)
ANSemCorrupt   = 81000008H; (* a semaphore is in illegal state *)
ANFreeTwice    = 81000009H; (* freeing memory already freed *)
ANBogusExcp    = 8100000AH; (* illegal 68k exception taken *)

(* graphics.library *)
ANGraphicsLib  = 02000000H;
ANGfxNoMem     = 82010000H; (* graphics out of memory *)
ANLongFrame    = 82010006H; (* long frame, no memory *)
ANShortFrame   = 82010007H; (* short frame, no memory *)
ANTextTmpRas   = 82010009H; (* text, no memory for TmpRas *)
ANBltBitMap    = 8201000AH; (* BltBitMap, no memory *)
ANRegionMemory = 8201000BH; (* regions, memory no available *)
ANMakeVPort    = 82010030H; (* MakeVPort, no memory *)
ANGfxNoLCM     = 82011234H; (* emergency memory not available *)

(* layers.library *)
ANLayersLib    = 03000000H;
ANLayersNoMem  = 83010000H; (* layers out of memory *)

(* intuition.library *)
ANIntuition    = 04000000H;
ANGadgetType   = 84000001H; (* unknown gadget type *)
ANBadGadget    = 04000001H; (* recovery form of ANGadgetType *)
ANCreatePort   = 84010002H; (* create port, no memory *)
ANItemAlloc    = 04010003H; (* item plane alloc, no memory *)
ANSubAlloc     = 04010004H; (* sub alloc, no memory *)
ANPlaneAlloc   = 84010005H; (* plane alloc, no memory *)
ANItemBoxTop   = 84000006H; (* item box top < RelZero *)
ANOpenScreen   = 84010007H; (* open screen, no memory *)
ANOpenScrnrast = 84010008H; (* open screen, raster alloc, no memory *)
ANSysScrnrType = 84000009H; (* open sys screen, unknown type *)
ANAddSWGadget  = 8401000AH; (* add SW gadgets, no memory *)
ANOpenWindow   = 8401000BH; (* open window, no memory *)
ANBadState     = 8400000CH; (* bad state return entering Intuition *)
ANBadMessage   = 8400000DH; (* bad message received by IDCMP *)
ANWeirdEcho    = 8400000EH; (* weird echo causing incomprehension *)
ANNoConsole    = 8400000FH; (* couldn't open the Console Device *)

(* math.library *)
ANMathLib      = 05000000H;

(* clist.library *)
ANCLstLib      = 06000000H;

(* dos.library *)
ANDOSLib       = 07000000H;
ANStartMem     = 07010001H; (* no memory at startup *)
ANEndTask      = 07000002H; (* EndTask didn't *)
ANQPKtFail     = 07000003H; (* Qpkt failure *)
ANAsyncPkt     = 07000004H; (* Unexpected packet received *)
ANFreeVec      = 07000005H; (* Freevec failed *)
ANDiskBlkSeq   = 07000006H; (* Disk block sequence error *)
ANBitMap       = 07000007H; (* Bitmap corrupt *)
ANKeyFree      = 07000008H; (* Key already free *)
ANBadChkSum    = 07000009H; (* Invalid checksum *)
ANDiskError    = 0700000AH; (* Disk Error *)
ANKeyRange     = 0700000BH; (* Key out of range *)
ANBadOverlay   = 0700000CH; (* Bad overlay *)

(* ramlib.library *)
ANRAMLib       = 08000000H;
ANBadSegList   = 08000001H; (* no overlays in library seglists *)
```



```

(* icon.library *)
ANIconLib = 09000000H;

(* expansion.library *)
ANExpansionLib = 0A000000H;
ANBadExpansionFree = 0A000001H;

(* audio.device *)
ANAUDIODev = 10000000H;

(* console.device *)
ANConsoleDev = 11000000H;

(* gameport.device *)
ANGamePortDev = 12000000H;

(* keyboard.device *)
ANKeyboardDev = 13000000H;

(* trackdisk.device *)
ANTrackDiskDev = 14000000H;
ANTDCalibSeek = 14000001H; (* calibrate: seek error *)
ANTDDelay = 14000002H; (* delay: error on timer wait *)

(* timer.device *)
ANTimerDev = 15000000H;
ANTMBadReq = 15000001H; (* bad request *)
ANTMBadSupply = 15000002H; (* power supply does no supply ticks *)

(* cia.resource *)
ANCIARsrc = 20000000H;

(* disk.resource *)
ANDiskRsrc = 21000000H;
ANDRHasDisk = 21000001H; (* get unit: already has disk *)
ANDRIntNoAct = 21000002H; (* interrupt: no active unit *)

(* misc.resource *)
ANMiscRsrc = 22000000H;

(* bootstrap *)
ANBootStrap = 30000000H;
ANBootError = 30000001H; (* boot code returned an error *)

(* Workbench *)
ANWorkbench = 31000000H;

(* DiskCopy *)
ANDiskCopy = 32000000H;

```

```

PROCEDURE Alert(alertNum : LONGCARD; parameters : LONGCARD);
(* Alert the user of an error.

```

```

    alertNum - a number indicating the particular alert
    parameters - additional parameters *)

```

```

END Alerts.

```

Module AmigaDOS

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: AmigaDOS.DEF      Version: Amiga.00.00
* Created: 11/18/86      Updated: 07/24/87      Author: Leon Frenkel
* Description: AmigaDOS types/functions.
*****)
```

DEFINITION MODULE AmigaDOS;

FROM SYSTEM IMPORT ADDRESS;
FROM System IMPORT DOSBase; (* Used by IMPLEMENTATION MODULE *)

CONST

DOSName = "dos.library";

(* Mode parameter to Open() *)

ModeReadWrite = 10040; (* Open old file with exclusive lock *)
ModeOldFile = 10050; (* Open existing file r/w pos at start of file *)
ModeNewFile = 10060; (* Open freshly created file (delete old file) r/w *)

(* Relative position to Seek() *)

OffsetBeginning = -10; (* relative to beginning of file *)
OffsetCurrent = 00; (* relative to current file position *)
OffsetEnd = 10; (* relative to end of file *)

(* Passed as type to Lock() *)

SharedLock = -20; (* File is readable by others *)
AccessRead = -20; (* Synonym *)
ExclusiveLock = -10; (* No other access allowed *)
AccessWrite = -10; (* Synonym *)

TYPE

(* All BCPL data must be long word aligned. BCPL pointers are the long word
* address (i.e. byte address divided by 4 (>>2) *)
BPTR = ADDRESS; (* long word pointer *)

(* BCPL strings have a length in the first byte and the the characters.

* for example: s[0]=3 s[1]=S s[2]=Y s[3]=S *)
BSTR = ADDRESS; (* long word pointer to BCPL string *)

FileHandle = BPTR; (* BCPL ptr to a file handle structure *)
FileLock = BPTR; (* BCPL ptr to a file lock structure *)

CONST

TicksPerSecond = 500; (* Number of ticks in one second *)

TYPE

(* DateStamp *)

DateStampPtr = POINTER TO DateStampRecord;
DateStampRecord = RECORD

 dsDays : LONGINT; (* Number of days since Jan. 1, 1978 *)
 dsMinute : LONGINT; (* Number of minutes past midnigth *)
 dsTick : LONGINT; (* Numbre of ticks past minute *)
END;

CONST

(* Definition of protection bits *)

FibArchive = 4; (* cleared whenever file is changed *)
FibRead = 3; (* ignored by system *)
FibWrite = 2; (* ignored by system *)
FibExecute = 1; (* ignored by system *)
FibDelete = 0; (* prevent file from being deleted *)

TYPE

```

ProtectionSet = SET OF [0..31];
(* Returned by Examine() and ExInfo(), must be on a 4 byte boundary *)
FileInfoBlockPtr = POINTER TO FileInfoBlock;
FileInfoBlock = RECORD
    fibDiskKey      : LONGINT;
    fibDirEntryType : LONGINT; (* <0 = File, >0 = Directory *)
    fibFileName     : ARRAY [0..107] OF CHAR; (* file name *)
    fibProtection   : ProtectionSet; (* file protection 3..0 rwxr *)
    fibEntryType    : LONGINT;
    fibSize         : LONGINT; (* Number of bytes in file *)
    fibNumBlocks    : LONGINT; (* Number of blocks in file *)
    fibDate         : DateStampRecord; (* Date file last changed *)
    fibComment      : ARRAY [0..115] OF CHAR; (* Comment str *)
END;

```

```
(* returned by Info(), must be on a 4 byte boundary *)
```

```
InfoDataPtr = POINTER TO InfoData;
```

```

InfoData = RECORD
    idNumSoftErrors : LONGINT;
    idUnitNumber    : LONGINT;
    idDiskState     : LONGINT;
    idNumBlocks     : LONGINT;
    idNumBlocksUsed : LONGINT;
    idBytesPerBlock : LONGINT;
    idDiskType      : LONGINT;
    idVolumeNode    : BPTR;
    idInUse         : LONGINT;
END;

```

CONST

```
(* Possible values of idDiskState *)
```

```
IDWriteProtected = 80D; (* Disk is write protected *)
```

```
IDValidating     = 81D; (* Disk is currently being validated *)
```

```
IDValidated      = 82D; (* Disk is consistent and writeable *)
```

```
(* Possible values of idDiskType *)
```

```
IDNoDiskPresent = -1D;
```

```
IDUnreadableDisk = 42414400H; (* 'BAD' *)
```

```
IDDosDisk        = 444F5300H; (* 'DOS' *)
```

```
IDNotReallyDos   = 4E444F53H; (* 'NDOS' *)
```

```
IDKickStartDisk = 4B494348H; (* 'KICK' *)
```

```
(* Errors from IoErr(), etc. *)
```

```
ErrorNoFreeStore = 103D;
```

```
ErrorTaskTableFull = 105D;
```

```
ErrorLineTooLong = 120D;
```

```
ErrorFileNotObject = 121D;
```

```
ErrorInvalidResidentLibrary = 122D;
```

```
ErrorNoDefaultDir = 201D;
```

```
ErrorObjectInUse = 202D;
```

```
ErrorObjectExists = 203D;
```

```
ErrorDirNotFound = 204D;
```

```
ErrorObjectNotFound = 205D;
```

```
ErrorBadStreamName = 206D;
```

```
ErrorObjectTooLarge = 207D;
```

```
ErrorActionNotKnown = 209D;
```

```
ErrorInvalidComponentName = 210D;
```

```
ErrorInvalidLock = 211D;
```

```
ErrorObjectWrongType = 212D;
```

```
ErrorDiskNotValidated = 213D;
```

```
ErrorDiskWriteProtected = 214D;
```

```
ErrorRenameAcrossDevices = 215D;
```

```
ErrorDirectoryNotEmpty = 216D;
```

```
ErrorTooManyLevels = 217D;
```

```
ErrorDeviceNotMounted = 218D;
```

```
ErrorSeekError = 219D;
```

```
ErrorCommentTooBit = 220D;
```

```
ErrorDiskFull = 221D;
```

```
ErrorDeleteProtected = 222D;
```

```
ErrorWriteProtected = 223D;
```

```
ErrorReadProtected = 224D;
```

```
ErrorNotADosDisk = 225D;
```

```
ErrorNoDisk = 226D;
```

```
ErrorNoMoreEntries = 232D;
```

Module AmigaDOS

```
(* These are the return codes used by convention by AmigaDOS commands *)
(* See FAILAT and IF for relevance to EXECUTE files *)
ReturnOk    = 00; (* no problems, success *)
ReturnWarn  = 50; (* a warning only *)
ReturnError = 100; (* somethin wrong *)
ReturnFail  = 200; (* complete or severe failure *)
```

```
(* Bit numbers that signal you that a user has issued a break *)
SigBreakCtrlC = 12;
SigBreakCtrlD = 13;
SigBreakCtrlE = 14;
SigBreakCtrlF = 15;
```

```
(* NOTE: AmigaDOS when returning a boolean result returns: *)
(* -1 (TRUE or SUCCESS), 0 (FALSE or FAILURE). *)
(* To be compatible with a Modula-2 Boolean value it is automatically *)
(* converted to 1 (TRUE or SUCCESS), 0 (FALSE or FAILURE) *)
```

```
(*****
(* File Handling Functions *)
*****)
```

```
PROCEDURE Close(file: FileHandle);
(* Close an open file.
```

file - file handle *)

```
PROCEDURE CreateDir(name: ADDRESS): FileLock;
(* Create a new director.
```

name - null terminated string

result - BCPL pointer to a lock *)

```
PROCEDURE CurrentDir(lock: FileLock): FileLock;
(* Make a directory associated with a lock the current working directory.
```

lock - BCPL pointer to a lock

result - BCPL pointer to a lock *)

```
PROCEDURE DeleteFile(name: ADDRESS): BOOLEAN;
(* Delete a file or director.
```

name - a null terminated string

result - success(TRUE) or failure(FALSE) *)

```
PROCEDURE DupLock(lock: FileLock): FileLock;
(* Duplicate a lock.
```

lock - BCPL pointer to a lock

result - BCPL pointer to a lock *)

```
PROCEDURE Examine(lock: FileLock; VAR infoBlock: FileInfoBlock): BOOLEAN;
(* Examine a directory or file associated with a lock.
```

lock - BCPL pointer to a lock

infoBlock - pointer to a file info block to be filled in

result - success(TRUE) or failure(FALSE) *)

```
PROCEDURE ExNext(lock: FileLock; VAR infoBlock: FileInfoBlock): BOOLEAN;
(* Examine the next entry in a directory.
```

```
lock - BCPL pointer to a lock
infoBlock - pointer to a file info block to be filled in
```

```
result - success(TRUE) or failure(FALSE) *)
```

```
PROCEDURE Info(lock: FileLock; VAR infoData: InfoData): BOOLEAN;
(* Return information about a disk.
```

```
lock - BCPL pointer to a lock
infoData - pointer to a InfoData structure to be filled in
```

```
result - success(TRUE) or failure(FALSE) *)
```

```
PROCEDURE Input(): FileHandle;
```

```
(* Return the initial input file handle assigned to the program.
```

```
result - file handle *)
```

```
PROCEDURE IoErr(): LONGINT;
```

```
(* Return extra information from the system after an error.
```

```
result - error code or other error info *)
```

```
PROCEDURE IsInteractive(file: FileHandle): BOOLEAN;
```

```
(* Test if a file is connected to a virtual terminal or not.
```

```
file - file handle
```

```
result - TRUE = interactive, FALSE = non-interactive *)
```

```
PROCEDURE Lock(name: ADDRESS; accessMode: LONGINT): FileLock;
```

```
(* Lock a directory or file.
```

```
name - a null terminated string
accessMode - SharedLock or ExclusiveLock
```

```
result - BCPL pointer to a lock *)
```

```
PROCEDURE Open(name: ADDRESS; accessMode: LONGINT): FileHandle;
```

```
(* Open a file for input or output.
```

```
name - a null terminated string
accessMode - ModeOldFile or ModeNewFile
```

```
result - file handle *)
```

```
PROCEDURE Output(): FileHandle;
```

```
(* Return the initial output file handle assigned to the program.
```

```
result - file handle *)
```

```
PROCEDURE ParentDir(lock: FileLock): FileLock;
```

```
(* Obtain the parent of a director or file.
```

```
lock - BCPL pointer to a lock
```

```
result - BCPL pointer to a lock *)
```

Module AmigaDOS

PROCEDURE Read(file: FileHandle; buffer: ADDRESS; length: LONGINT): LONGINT;
(* Read n bytes of data from a file.

file - file handle
buffer - pointer to a buffer
length - number of bytes to read

result - actual number of bytes read or -1 if an error occurred *)

PROCEDURE Rename(oldName, newName: ADDRESS): BOOLEAN;
(* Rename a directory or file.

oldName - null terminated string
newName - null terminated string

result - success(TRUE) or failure(FALSE) *)

PROCEDURE Seek(file: FileHandle; position: LONGINT; mode: LONGINT): LONGINT;
(* Move to a logical file position.

file - file handle
position - an offset to move file position ptr to, positive or negative
mode - OffsetBeginning or OffsetCurrent or OffsetEnd

result - old file position, or -1 if an error occurred *)

PROCEDURE SetComment(name, comment: ADDRESS): BOOLEAN;
(* Set a comment on file or directory.

name - null terminated string
comment - null terminated string of up to 80 characters in length

result - success(TRUE) or failure(FALSE) *)

PROCEDURE SetProtection(name: ADDRESS; mask: ProtectionSet): BOOLEAN;
(* Set file or directory protection.

name - null terminated string
mask - protection make to be set

result - success(TRUE) or failure(FALSE) *)

PROCEDURE UnLock(lock: FileLock);
(* Unlock a directory or file.

lock - BCPL pointer to a lock *)

PROCEDURE WaitForChar(file: FileHandle; timeout: LONGINT): BOOLEAN;
(* Indicates whether characters arrive within a time limit or not.

file - file handle
timeout - amount of time to wait in micro seconds

result - if character arrived before timeout returns TRUE, else FALSE *)

PROCEDURE Write(file: FileHandle; buffer: ADDRESS; length: LONGINT): LONGINT;
(* Write n bytes of data to a file.

file - file handle
buffer - pointer to a buffer
length - number of bytes to write

result - actual number of bytes written, or -1 if an error occurred *)

```

(*****)
(*      Loader Functions      *)
(*****)

```

```

PROCEDURE Execute(cmdString: ADDRESS; input, output: FileHandle): BOOLEAN;
(* Execute a CLI command.

```

```

    cmdString - null terminated string
    input - file handle
    output - file handle

```

```

    result - success(TRUE) or failure(FALSE) *)

```

```

PROCEDURE LoadSeg(name: ADDRESS): BPTR;
(* Load a load module into memory.

```

```

    name - null terminated string

    result - pointer to a segment or 0 if the load failed *)

```

```

PROCEDURE UnLoadSeg(segment: BPTR);
(* Unload a segment previously loaded by LoadSeg.

```

```

    segment - pointer to a segment originally returned by LoadSeg *)

```

```

(*****)
(*      Misc Functions      *)
(*****)

```

```

PROCEDURE DateStamp(VAR v: DateStampRecord);
(* Obtain the current date and time in internal format.

```

```

    v - pointer to a DateStamp structure (3 Long Words *)

```

```

END AmigaDOS.

```


Module AmigaDOSExt

```
(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: AmigaDOSExt.DEF      Version: Amiga.00.00
* Created: 12/03/86      Updated: 07/24/87      Author: Leon Frenkel
* Description: AmigaDOS types which are not needed by casual DOS users.
*****)
```

DEFINITION MODULE AmigaDOSExt;

```
FROM SYSTEM IMPORT ADDRESS;
FROM AmigaDOS IMPORT BPTR, BSTR, DateStampRecord;
FROM Libraries IMPORT Library;
FROM Ports IMPORT MessagePtr, MsgPort, MsgPortPtr;
```

TYPE

```
(* The BPTR of this structure is returned by Open() and other routines
* that return a file. You need only worry about this structure to do
* async io's via PutMsg() instead of standard file system calls
*)
```

```
FileHandlePtr = POINTER TO FileHandleRecord;
FileHandleRecord = RECORD
```

```
  fhLink : MessagePtr; (* Exec Message *)
  fhPort : MsgPortPtr; (* Reply port for the packet *)
  fhType : MsgPortPtr; (* Port to do PutMsg() to *)
  fhBuf : BPTR; (* Adr is negative if a plain file *)
  fhPos : LONGINT;
  fhEnd : LONGINT;
  fhFuncs : ADDRESS;
  fhFunc2 : ADDRESS;
  fhFunc3 : ADDRESS;
  fhArgs : LONGINT;
  fhArg2 : LONGINT;
```

END;

```
(* a lock structure, as returned by Lock() or DupLock() *)
```

```
FileLockPtr = POINTER TO FileLockRecord;
FileLockRecord = RECORD
```

```
  flLink : BPTR; (* bcpl pointer to next lock *)
  flKey : LONGINT; (* disk block number *)
  flAccess : LONGINT; (* exclusive or shared *)
  flTask : MsgPortPtr; (* handler task's port *)
  flVolume : BPTR; (* bcpl pointer to a DeviceList *)
```

END;

```
RootNodePtr = POINTER TO RootNode;
```

```
RootNode = RECORD
```

```
  rnTaskArray : BPTR; (* [0] is max number of CLI's
                       * [1] is APTR to processid of CLI 1
                       * [n] is APTR to processid of CLI n *)
  rnConsoleSegment : BPTR; (* SegList for the CLI *)
  rnTime : DateStampRecord; (* Current time *)
  rnRestartSeg : BPTR; (* SegList for the disk validator process *)
  rnInfo : BPTR; (* ptr to the Info structure *)
  rnFileHandlerSegment : BPTR; (* SegList for a file handler *)
```

END;

```
(* DOS library node structure.
```

```
* This is the data at positive offsets from the library node. Negative
* offsets from the node is the jump table to DOS functions. *)
```

```
DosLibraryPtr = POINTER TO DosLibrary;
```

```
DosLibrary = RECORD
```

```
  dlLib : Library;
  dlRoot : RootNodePtr;
  dlGV : ADDRESS; (* pointer to BCPL global vector *)
  dlA2 : ADDRESS; (* private register dump of DOS *)
  dlA5 : ADDRESS;
  dlA6 : ADDRESS;
```

END;


```

DosInfoPtr = POINTER TO DosInfo;
DosInfo = RECORD
    diMcName      : BPTR;      (* Network name of this machine; currently 0 *)
    diDevInfo     : BPTR;      (* Device List *)
    diDevices     : BPTR;      (* Currently zero *)
    diHandlers    : BPTR;      (* Currently zero *)
    diNetHand     : ADDRESS;    (* Network handler processid; currently 0 *)
END;

```

(* DOS Processes started from the CLI via RUN or NEWCLI have this additional
* set of data associated with them *)

```

CommandLineInterfacePtr = POINTER TO CommandLineInterface;

```

```

CommandLineInterface =

```

```

RECORD

```

```

    cliResult2      : LONGINT; (* Value of IoErr from last command *)
    cliSetName      : BSTR;     (* Name of current directory *)
    cliCommandDir    : BPTR;     (* Lock associated with command dir *)
    cliReturnCode    : LONGINT; (* Return code from last command *)
    cliCommandName   : BSTR;     (* Name of current command *)
    cliFailLevel     : LONGINT; (* Fail level (set by FAILAT) *)
    cliPrompt        : BSTR;     (* Current prompt (set by PROMPT) *)
    cliStandardInput : BPTR;     (* Default (terminal) CLI input *)
    cliCurrentInput  : BPTR;     (* Current CLI input *)
    cliCommandFile   : BSTR;     (* Name of EXECUTE command file *)
    cliInteractive   : LONGINT;  (* Boolean: True if prompts required *)
    cliBackground    : LONGINT;  (* Boolean: True if CLI created by RUN *)
    cliCurrentOutput : BPTR;     (* Current CLI output *)
    cliDefaultStack  : LONGINT;  (* Stack size to be obtained in lwords *)
    cliStandardOutput : BPTR;    (* Default (terminal) CLI output *)
    cliModule        : BPTR;     (* SegList of currently loaded command *)
END;

```

```

CONST

```

(* definitions for dType field of DeviceList record *)

```

DLTDevice = 0D;

```

```

DLTDirectory = 1D;

```

```

DLTVolume = 2D;

```

```

TYPE

```

(* Note From dosextens.h file from which this structure was taken.
* This structure needs some work. It should really be a union, because
* it can take on different values depending on whether it is a device,
* an assigned directory, or a volume. For now it reflects a volume.
*)

```

DeviceListPtr = POINTER TO DeviceList;

```

```

DeviceList = RECORD

```

```

    dlNext      : BPTR;      (* bptr to next device list *)
    dlType      : LONGINT;
    dlTask      : MsgPortPtr; (* ptr to handler task *)
    dlLock      : BPTR;      (* not for volumes *)
    dlVolumeDate : DateStampRecord; (* creation date *)
    dlLockList   : BPTR;      (* outstanding locks *)
    dlDiskType   : LONGINT;    (* 'DOS', etc. *)
    dlUnused     : LONGINT;
    dlName      : BSTR;      (* bcp1 to name *)
END;

```

```

END AmigaDOSExt.

```

Module AmigaDOSProcess

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: AmigaDOSProcess.DEF      Version: Amiga.00.00      *
* Created: 11/18/86   Updated: 07/24/87   Author: Leon Frenkel      *
* Description: AmigaDOS Process related types/functions.      *
*****)
```

DEFINITION MODULE AmigaDOSProcess;

```
FROM SYSTEM IMPORT ADDRESS;
FROM System IMPORT DOSBase; (* Used by IMPLEMENTATION MODULE *)
FROM AmigaDOS IMPORT BPTR;
FROM Ports IMPORT Message, MessagePtr, MsgPort, MsgPortPtr;
FROM Tasks IMPORT Task;
```

TYPE

```
(* All DOS processes have this structure *)
(* Create and DeviceProc returns pointer to the MsgPort in this structure *)
(* To get a pointer to the process the following assignment can be done: *)
(* DevProc = ProcessPtr(DeviceProc(...)-ADDRESS(TSIZE(Task))); *)
```

ProcessID = ADDRESS; (* process identifier as returned by CreateProc() *)

ProcessPtr = POINTER TO Process;

Process = RECORD

```
  prTask      : Task;      (* Exec Task structure *)
  prMsgPort    : MsgPort;  (* BPTR address from DOS functions *)
  prPad        : INTEGER;  (* Remaining variables on 4 byte boundaries *)
  prSegList    : BPTR;     (* Array of seg lists used by this process *)
  prStackSize  : LONGINT;  (* Size of process stack in bytes *)
  prGlobVec    : ADDRESS;  (* Global vector for this process *)
  prTaskNum    : LONGINT;  (* CLI task number or zero if a CLI *)
  prStackBase  : BPTR;     (* Ptr to high memory end of stack *)
  prResult2    : LONGINT;  (* Value of 2nd result from last call *)
  prCurrentDir : BPTR;     (* Lock associated with current dir *)
  prCIS        : BPTR;     (* Current CLI Input Stream *)
  prCOS        : BPTR;     (* Current CLI Output Stream *)
  prConsoleTask : ADDRESS; (* Console handler process for current window *)
  prFileSystemTask : ADDRESS; (* File handler process for current dirve *)
  prCLI        : BPTR;     (* pointer to ConsoleLineInterpreter *)
  prReturnAddr : ADDRESS;  (* pointer to previous stack frame *)
  prPktWait    : ADDRESS;  (* function to be called when awaiting msg *)
  prWindowPtr  : ADDRESS;  (* Window for error printing *)
```

END;

(* This is the extension to EXEC Messages used by DOS *)

DosPacketPtr = POINTER TO DosPacket;

DosPacket = RECORD

```
  dpLink : MessagePtr; (* Exec message *)
  dpPort : MsgPortPtr; (* Reply port for the packet
                        * must be filled in each send *)
  dpType : LONGINT;    (* See Action... below and
                        * 'R' means Read, 'W' means Write to the
                        * file system *)
  dpRes1 : LONGINT;    (* For file system calls this is what would
                        * have been returned by IoErr() *)
  dpRes2 : LONGINT;    (* For file system calls this is what would
                        * have been returned by IoErr() *)
  dpArg1 : LONGINT;
  dpArg2 : LONGINT;
  dpArg3 : LONGINT;
  dpArg4 : LONGINT;
  dpArg5 : LONGINT;
  dpArg6 : LONGINT;
  dpArg7 : LONGINT;
```

END;

(* A Packet does not require the Message to be before it in memory, but

* for convenience it is useful to associate the two.

*)

```

StandardPacketPtr = POINTER TO StandardPacket;
StandardPacket = RECORD
    spMsg : Message;
    spPkt : DosPacket;
END;

```

```
CONST
```

```
(* Packet Types *)
```

```

ActionNIL           = 00;
ActionGetBlock      = 20;
ActionSetMap        = 40;
ActionDie           = 50;
ActionEvent         = 60;
ActionCurrentVolume = 70;
ActionLocateObject  = 80;
ActionRenameDisk    = 90;
ActionWrite         = LONGINT('W');
ActionRead          = LONGINT('R');
ActionFreeLock      = 150;
ActionDeleteObject  = 160;
ActionRenameObject  = 170;
ActionCopyDir       = 190;
ActionWaitChar      = 200;
ActionSetProtect    = 210;
ActionCreateDir     = 220;
ActionExamineObject = 230;
ActionExamineNext   = 240;
ActionDiskInfo      = 250;
ActionInfo          = 260;
ActionSetComment    = 280;
ActionParent        = 290;
ActionTimer         = 300;
ActionInhibit       = 310;
ActionDiskType      = 320;
ActionDiskChange    = 330;
ActionSetFileDate   = 340;
ActionSetRawMode    = 9940;

```

```

PROCEDURE CreateProc(name: ADDRESS; pri: LONGINT;
    segment: ADDRESS; stackSize: LONGINT): ProcessID;
(* Create a new process.

```

```

    name - null terminated string
    pri - priority to be given to process
    segment - pointer to segment list, as returned by LoadSeg()
    stackSize - size of root stack in bytes

```

```
    result - the ProcessID or 0 if an error occurred *)
```

```

PROCEDURE Delay(ticks: LONGINT);
(* Delay a process for a specified amount of time.

```

```
    ticks - number of ticks to wait (50 ticks/second) *)
```

```

PROCEDURE DeviceProc(name: ADDRESS): ProcessID;
(* Return the process identifier of the process handling that I/O.

```

```
    name - null terminated string
```

```

    result - process identifier of the device handler or 0 if not found.
    If the name refers to a file on a mounted device then a
    pointer to a directory lock is returned by IoErr() *)

```

```

PROCEDURE Exit(returnCode: LONGINT);
(* Exit from a process.

```

```

    returnCode - if the process is running under the CLI this will be the
    return code returned to the CLI. If the program is running
    as a distinct process, Exit deletes the process and releases
    the space associated with the stack, segment list, and process
    structure *)

```

Module AmigaDOSProcess

PROCEDURE GetPacket(wait: LONGINT): DosPacketPtr;
(* Returns a packet if one is ready.

wait - if (-1) then wait for a packet if not ready. if (0) then don't
wait if packet is not ready

result - pointer to packet, 0 if no packet was ready *)

PROCEDURE QueuePacket(VAR packet: DosPacket): LONGINT;
(* Queue a packet.

packet - a dos packet

result - successful(-1) or fail(0) ??? *)

END AmigaDOSProcess.

```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: Areas.DEF                               Version: Amiga.00.00                       *
* Created: 11/20/86   Updated: 05/08/87   Author: Leon Frenkel                         *
* Description: Graphics Areas types/functions.                                           *
*****)

```

DEFINITION MODULE Areas;

```

FROM SYSTEM IMPORT ADDRESS;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Rasters IMPORT RastPort;

```

TYPE

```

  AreaInfoPtr = POINTER TO AreaInfo;
  AreaInfo = RECORD
    VctrTbl : ADDRESS; (* ptr to start of vector table *)
    VctrPtr : ADDRESS; (* ptr to current vertex *)
    FlagTbl : ADDRESS; (* ptr to start of vector flag table *)
    FlagPtr : ADDRESS; (* ptrs to areafill flags *)
    Count : INTEGER; (* number of vertices in list *)
    MaxCount : INTEGER; (* AreaMove/Draw will not allow Count>MaxCount *)
    FirstX : INTEGER; (* first point for this polygon *)
    FirstY : INTEGER;
  END;

```

PROCEDURE AreaCircle(VAR rp: RastPort; cx, cy, radius: INTEGER): INTEGER;

(* Add a circle to areainfo list for areafill. This function corresponds
* to the 'C' language macro. The c macro is:

```
#define AreaCircle(rp,cx,cy,r) AreaEllipse(rp,cx,cy,r,r);
```

```

rp - a RastPort structure
cx - x part of "centerpoint" coordinate in raster
cy - y part of "centerpoint" coordinate in raster
radius - radius of circle

```

result - 0 = no error, -1 = no space left in vector list *)

PROCEDURE AreaDraw(VAR rp: RastPort; x, y: INTEGER): INTEGER;

(* Add a point to a list of end points for areafill.

```

rp - RastPort structure
x - x part of coordinate of a point in the raster
y - y part of coordinate of a point in the raster

```

result - 0 = no error, -1 = no space left in vector list *)

PROCEDURE AreaEllipse(VAR rp: RastPort; cx, cy, a, b: INTEGER): INTEGER;

(* Add an ellipse to areainfo list for areafill.

```

rp - a RastPort structure
cx - x part of "centerpoint" coordinate in raster
cy - y part of "centerpoint" coordinate in raster
a - the horizontal radius of the ellipse (note: a must be > 0)
b - the vertical radius of the ellipse (note: b must be > 0)

```

result - 0 = no error, -1 = no space left in vector list *)

PROCEDURE AreaEnd(VAR rp: RastPort): INTEGER;

(* Process table of vectors and produce areafill.

```
rp - RastPort structure
```

result - 0 = no error, -1 = error occurred *)

Module Areas

```
PROCEDURE AreaMove(VAR rp: RastPort; x, y: INTEGER): INTEGER;
(* Define a new starting point for a new shape in the vector list.
```

rp - RastPort structure

x - x part of coordinate of a point in the raster

y - y part of coordinate of a point in the raster

result - 0 = no error, -1 = no space left in vector list *)

```
PROCEDURE InitArea(VAR areaInfo: AreaInfo; buffer: ADDRESS; maxvects: INTEGER);
```

```
(* Initialize vector collection matrix.
```

areaInfo - AreaInfo structure

buffer - pointer to chunk of memory to collect vertices

maxvectors - max number of vectors thsi buffer can hold *)

END Areas.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1987 by Leon Frenkel                *
*                               *
*****
* Name: ASCII.DEF                               Version: Amiga.00.00
* Created: 07/22/87   Updated: 07/24/87   Author: Leon Frenkel
* Description: ASCII codes and functions.
*****
)
```

DEFINITION MODULE ASCII;

(* \$L *)

CONST

```

NUL = 00C; SOH = 01C; STX = 02C; ETX = 03C; EOT = 04C; ENQ = 05C;
ACK = 06C; BEL = 07C; BS = 08C; HT = 09C; LF = 0AC; VT = 0BC;
FF = 0CC; CR = 0DC; SO = 0EC; SI = 0FC; DLE = 10C; DC1 = 11C;
DC2 = 12C; DC3 = 13C; DC4 = 14C; NAK = 15C; SYN = 16C; ETB = 17C;
CAN = 18C; EM = 19C; SUB = 1AC; ESC = 1BC; FS = 1CC; GS = 1DC;
RS = 1EC; US = 1FC; DEL = 177C;
```

```

OctalChars      = '01234567';
DecimalChars    = '0123456789';
HexadecimalChars = '0123456789ABCDEF';
AlphabetChars    = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
RealChars       = '0123456789.E+-';
```

TYPE

```
CharCases = (Upper, Lower, UpperOrLower, Shift, Control, ControlShift);
```

```
PROCEDURE CharIsPrintable(Ch: CHAR): BOOLEAN;
(* Returns TRUE if Ch is printable and ASCII *)
```

```
PROCEDURE CharIsControl(Ch: CHAR): BOOLEAN;
(* Returns TRUE if Ch is ASCII control char *)
```

```
PROCEDURE CharIsASCII(Ch: CHAR): BOOLEAN;
(* Returns TRUE if (CharIsPrintable OR CharIsControl) *)
```

END ASCII.

Module AudioDevice

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: AudioDevice.DEF                      Version: Amiga.00.00      *
* Created: 11/17/86   Updated: 05/07/87   Author: Leon Frenkel        *
* Description: Audio Device.                                          *
*****)
```

DEFINITION MODULE AudioDevice;

FROM SYSTEM IMPORT ADDRESS;
FROM IODevices IMPORT CmdNonStd, IORequest;
FROM Ports IMPORT Message;

CONST

AudioName = "audio.device";

(* Audio Channels *)

Left0 = 0;
Right0 = 1;
Right1 = 2;
Left1 = 3;

TYPE

AudioChannelsSet = SET OF [Left0..Left1];

CONST

ADHardChannels = 4;

ADAllocMinPrec = -128;

ADAllocMaxPrec = 127;

ADCmdFree = CmdNonStd + 0;

ADCmdSetPrec = CmdNonStd + 1;

ADCmdFinish = CmdNonStd + 2;

ADCmdPerVol = CmdNonStd + 3;

ADCmdLock = CmdNonStd + 4;

ADCmdWaitCycle = CmdNonStd + 5;

ADCmdNoUnit = 20H; (* ADCMF_NOUNIT (1<<5) *)

ADCmdAllocate = 20H; (* ADCMD_ALLOCATE (ADCMF_NOUNIT+0) *)

(* .ioFlags := IOFlagsSet{ } *)

ADIOPerVol = 4;

ADIOSyncCycle = 5;

ADIONoWait = 6;

ADIOWriteMessage = 7;

ADIOErrNoAllocation = -10;

ADIOErrAllocFailed = -11;

ADIOErrChannelStolen = -12;

TYPE

IOAudioPtr = POINTER TO IOAudio;

IOAudio = RECORD

ioaRequest : IORequest;

ioaAllocKey : INTEGER;

ioaData : ADDRESS;

ioaLength : LONGCARD;

ioaPeriod : CARDINAL;

ioaVolume : CARDINAL;

ioaCycles : CARDINAL;

ioaWriteMsg : Message;

END;

END AudioDevice.


```

(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: Blit.DEF
* Created: 11/24/86   Updated: 07/24/87   Author: Leon Frenkel
* Description: Graphics Blitter types/functions.
*****)

```

```

DEFINITION MODULE Blit;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Graphics IMPORT BitMap;
FROM Rasters IMPORT RastPort;

```

```

CONST

```

```

  HSizeBits = 6;
  VSizeBits = 16 - HSizeBits;
  HSizeMask = 3FH; (* 2^6 - 1 *)
  VSizeMask = 3FFH; (* 2^10 - 1 *)

```

```

  MaxBytesPerRow = 128;

```

```

  (* definitions for blitter control register 0 *)

```

```

  NANBNC = 0;
  NANBC = 1;
  NABNC = 2;
  NABC = 3;
  ANBNC = 4;
  ANBC = 5;
  ABNC = 6;
  ABC = 7;
  Dest = 8;
  SrcC = 9;
  SrcB = 10;
  SrcA = 11;

```

```

  (* some commonly used operations *)

```

```

  AorB = {ABC, ANBC, NABC, ABNC, ANBNC, NABNC};
  AorC = {ABC, NABC, ABNC, ANBC, NANBC, ANBNC};
  AxorC = {NABC, ABNC, NANBC, ANBNC};
  AtoD = {ABC, ANBC, ABNC, ANBNC};

```

```

  AShiftShift = 12; (* bits to right align ashift value *)

```

```

  BShiftShift = 12; (* bits to right align bshift value *)

```

```

  (* definitions for blitter control register 1 bits *)

```

```

  LineMode = 0;
  OneDot = 1; (* one dot per horizontal line *)
  FillCarryIn = 2;
  FillOr = 3;
  FillXor = 4;
  OvFlag = 5;
  SignFlag = 6;
  BltReverse = OneDot;

```

```

  SUD = 4;

```

```

  SUL = 3;

```

```

  AUL = 2;

```

```

  Octant8 = {SUD, SUL};
  Octant7 = {AUL};
  Octant6 = {SUL, AUL};
  Octant5 = {SUD, SUL, AUL};
  Octant4 = {SUD, AUL};
  Octant3 = {SUL};
  Octant2 = {};
  Octant1 = {SUD};

```

Module Blit

TYPE

```
(* stuff for blit queuer *)
bltnodeptr = POINTER TO bltnode;
bltnode = RECORD
    n          : bltnodeptr;
    function   : ADDRESS;
    stat       : BYTE;
    blitsize   : INTEGER;
    beamsync   : INTEGER;
    cleanup    : ADDRESS;
END;
```

CONST

```
(* defined bits for bltstat *)
CleanUp = 40H;
CleanMe = CleanUp;
```

TYPE

```
(* Flags type for BltClear() function *)
BltClearFlagsSet = SET OF [0..31];
```

CONST

```
(* Flag definitions for BltClear function *)
BltClearWait = 0; (* force func to wait until blit is done *)
BltClearXY = 1; (* use row/bytesperrow *)
```

TYPE

```
MinTermFlagsSet = SET OF [0..7];
```

```
PROCEDURE BltBitMap(VAR SrcBitMap: BitMap; SrcX, SrcY: INTEGER;
VAR DstBitMap: BitMap; DstX, DstY: INTEGER;
SizeX, SizeY: INTEGER; Minterm: MinTermFlagsSet;
Mask: BYTE; TempA: ADDRESS): LONGCARD;
(* Move a rectangular region of bits in a BitMap.
```

SrcBitMap - the source bitmap structure
 SrcX - the x component of the upper left corner of the source rectangle
 SrcY - the y component of the upper left corner of the source rectangle
 DstBitMap - the destination bitmap structure
 DstX - the x component of the upper left corner of the dest rectangle
 DstY - the y component of the upper left corner of the dest rectangle
 SizeX - the width of the rectangle to be moved
 SizeY - the height of the rectangle to be moved
 Minterm - the logic function to apply to the rectangle
 Mask - the write mask to apply to this operation
 TempA - Ptr to chip memory for one source line or NIL

result - the number of planes actually involved in the blit *)

```
PROCEDURE BltBitMapRastPort(VAR srcbm: BitMap; srcX, srcY: INTEGER;
VAR dstp: RastPort; destX, destY: INTEGER;
sizeX, sizeY: INTEGER; minterm: MinTermFlagsSet);
(* Blit from source bitmap to destination rastport.
```

srcbm - a pointer to the source bitmap
 srcx - x offset into source bitmap
 srcy - y offset into source bitmap
 dstp - a pointer to the destination rastport
 destX - x offset into dest rastport
 destY - y offset into dest rastport
 sizeX - width of blit in pixels
 sizeY - height of blit in rows
 minterm - minterm to use for this blit *)

```
PROCEDURE BltClear(memBlock: ADDRESS; bytecount: LONGCARD;
                  flags: BltClearFlagsSet);
```

(* Clear a block of memory words to zero.

memBlock - pointer to local memory to be cleared
 bytecount - if bit 1 of flags = 0 then even number of bytes to clear, else
 low 16-bits is taken as number of bytes per row and upper 16-bits
 taken as number of rows
 flags - set bit 0 to force function to wait until blit is done
 set bit 1 to use row/bytesperrow *)

```
PROCEDURE BltMaskBitMapRastPort(VAR srcbm: BitMap; srcx, srcy: INTEGER;
                                VAR dstRp: RastPort; destX, destY: INTEGER;
                                sizeX, sizeY: INTEGER; minterm: MinTermFlagsSet;
                                bltmask : ADDRESS);
```

(* Blit from source bitmap to dest bitmap rastport with masking of source image.

srcbm - a pointer to the source bitmap
 srcx - x offset into source bitmap
 srcy - y offset into source bitmap
 dstRp - a pointer to the destination rastport
 destX - x offset into dest rastport
 destY - y offset into dest rastport
 sizeX - width of blit in pixels
 sizeY - height of blit in rows
 minterm - either (ABC|ABNC|ANBC) if copy source and blit thru mask
 or (ANBC) if invert source and blit thru mask
 bltmask - pointer to the single bit-plane mask, which must be the same size
 and dimension as the planes of the source bitmap *)

```
PROCEDURE BltPattern(VAR rp: RastPort; mask: ADDRESS;
                    x1, y1, maxx, maxy : INTEGER; bytecnt : INTEGER);
```

(* Using standard drawing rules for areafill, blit through a mask.

rp - point to RastPort
 mask - points to 2 dimensional mask or if NIL then use a rectangle
 x1 - x of upper left of rectangular region in RastPort
 y1 - y of upper left of rectangular region in RastPort
 maxx - x of pointer to lower right of rectangular region in RastPort
 maxy - y of pointer to lower right of rectangular region in RastPort
 bytecnt - BytesPerRow for mask *)

```
PROCEDURE BltTemplate(SrcTemplate: ADDRESS; SrcX: INTEGER; SrcMod: INTEGER;
                     VAR rp: RastPort; DstX, DstY: INTEGER;
                     SizeX, SizeY: INTEGER);
```

(* Cookie cut a shape in a rectangle to the RastPort.

SrcTemplate - pointer to the first (nearest) word of the template mask
 SrcX - x bit offset into the template mask (range 0..15)
 SrcMod - number of bytes per row in template mask
 rp - pointer to destination RastPort
 DstX - x of coordinate of the upper left corner of the destination for the blit
 DstY - y of coordinate of the upper left corner of the destination for the blit
 SizeX - x size of the rectangle to be used as the template
 SizeY - y size of the rectangle to be used as the template *)

Module Blit

```
PROCEDURE ClipBlit(VAR Src: RastPort; SrcX, SrcY: INTEGER;  
VAR Dest: RastPort; DestX, DestY: INTEGER;  
XSize, YSize: INTEGER; Minterm: MinTermFlagsSet);
```

(* Calls BltBitMap() after accounting for windows.

Src - RastPort of the source for your blit
SrcX - x component of the topleft offset into Src for your data
SrcY - y component of the topleft offset into Src for your data
Dest - RastPort to receive the blitted data
DestX - x component of the topleft offset into destination RastPort
DestY - y component of the topleft offset into destination RastPort
XSize - the width of the blit
YSize - the height of the blit
Minterm - the boolean blitter function, where SRCB is associated with the
Src RastPort and SRCC goes to the Dest RastPort *)

```
PROCEDURE DisownBlitter;  
(* Return blitter to free state. *)
```

```
PROCEDURE OwnBlitter;  
(* Get the blitter for private usage. *)
```

```
PROCEDURE QBlit(VAR bp: bltnode);  
(* Queue up a request for blitter usage.
```

bp - a blit structure *)

```
PROCEDURE QBSBlit(VAR bsp: bltnode);  
(* Synchronize the blitter request with the video beam.
```

bsp - a blit structure. *)

```
PROCEDURE VBeamPos(): LONGINT;  
(* Get vertical beam position at this instant.
```

result - current beam position in the range of 0..511 *)

```
PROCEDURE WaitBlit;  
(* Wait for the blitter to be finished before proceeding with anything else. *)
```

```
PROCEDURE WaitTOF;  
(* Wait for the top of the next video frame. *)
```

END Blit.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: BootBlock.DEF                      Version: Amiga.00.00 *
* Created: 11/26/86   Updated: 11/26/86   Author: Leon Frenkel *
* Description: Disk Boot Block types.          *
*****)

```

```

DEFINITION MODULE BootBlock;

```

```

TYPE

```

```

  BootBlockPtr = POINTER TO BootBlock;
  BootBlock = RECORD
    bbid      : LONGCARD; (* 4 char ideinifier *)
    bbchksum  : LONGINT;  (* boot block checksum (balance) *)
    bbdosblock : LONGINT;  (* reserved for DOS patch *)
  END;

```

```

CONST

```

```

  BootSects = 2; (* 1K bootstrap *)

  BBNameDos = 444F5300H; (* 'DOS' *)
  BBNameKick = 4B49434BH; (* 'KICK' *)

```

```

END BootBlock.

```

Module CIAHardware

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: CIAHardware.DEF                      Version: Amiga.00.00      *
* Created: 11/27/86   Updated: 05/05/87   Author: Leon Frenkel        *
* Description: Amiga Complex Interface Adapter definitions.            *
*****)
```

DEFINITION MODULE CIAHardware;

FROM SYSTEM IMPORT BYTE;

(* \$L+ *)

TYPE

CIAPAD = ARRAY [0..253] OF BYTE; (* 254 bytes (also causes word alignment) *)
 CIABITS = SET OF [0..7];

CIAPtr = POINTER TO CIA;

CIA = RECORD

```
  ciapra   : CIABITS;
  pad0     : CIAPAD;
  ciaprb   : CIABITS;
  pad1     : CIAPAD;
  ciaddra  : CIABITS;
  pad2     : CIAPAD;
  ciaddrb  : CIABITS;
  pad3     : CIAPAD;
  ciatalo  : CIABITS;
  pad4     : CIAPAD;
  ciatahi  : CIABITS;
  pad5     : CIAPAD;
  ciatblo  : CIABITS;
  pad6     : CIAPAD;
  ciatbhi  : CIABITS;
  pad7     : CIAPAD;
  ciatodlow : CIABITS;
  pad8     : CIAPAD;
  ciatodmid : CIABITS;
  pad9     : CIAPAD;
  ciatodhi  : CIABITS;
  pad10    : CIAPAD;
  unusedreg : CIABITS;
  pad11    : CIAPAD;
  ciasdr   : CIABITS;
  pad12    : CIAPAD;
  ciaicr   : CIABITS;
  pad13    : CIAPAD;
  ciacra   : CIABITS;
  pad14    : CIAPAD;
  ciacrb   : CIABITS;
```

END;

CONST

(* interrupt control register bit numbers *)

```
CIAicrTA = 0;
CIAicrTB = 1;
CIAicrALRM = 2;
CIAicrSP = 3;
CIAicrFLG = 4;
CIAicrIR = 7;
CIAicrSETCLR = 7;
```

(* control register A bit numbers *)

```
CIACraSTART = 0;
CIACraPBON = 1;
CIACraOUTMODE = 2;
CIACraRUNMODE = 3;
CIACraLOAD = 4;
CIACraINMODE = 5;
CIACraSPMODE = 6;
CIACraTODIN = 7;
```

```

(* control register B bit numbers *)
CIAcrbSTART = 0;
CIAcrbPBON = 1;
CIAcrbOUTMODE = 2;
CIAcrbRUNMODE = 3;
CIAcrbLOAD = 4;
CIAcrbINMODE0 = 5;
CIAcrbINMODE1 = 6;
CIAcrbALARM = 7;

(* control register B INMODE masks *)
CIAcrbInPHI2 = CIABITS{};
CIAcrbInCNT = CIABITS{CIAcrbINMODE0};
CIAcrbInTA = CIABITS{CIAcrbINMODE1};
CIAcrbInNTTA = CIABITS{CIAcrbINMODE0, CIAcrbINMODE1};

(*****
(* Port definitions - what each bit in a cia peripheral register is tied to *)
(*****

(* ciaa port A (0xbfe001) *)
CIAGamePort1 = 7; (* gameport 1, pin 6 (fire button) *)
CIAGamePort0 = 6; (* gameport 0, pin 6 (fire button) *)
CIADskRdy = 5; (* disk ready *)
CIADskTrack0 = 4; (* disk on track 00 *)
CIADskProt = 3; (* disk write protect *)
CIADskChange = 2; (* disk change *)
CIALED = 1; (* led light control (0==>bright) *)
CIAOverlay = 0; (* memory overlay bit *)

(* ciaa port B (0xbfe101) -- parallel port *)

(* ciab port A (0xbfd000) -- serial and printer control *)
CIAComDTR = 7; (* serial Data Terminal Ready *)
CIAComRTS = 6; (* serial Request to Send *)
CIAComCD = 5; (* serial Carrier Detect *)
CIAComCTS = 4; (* serial Clear to Send *)
CIAComDSR = 3; (* serial Data Set Ready *)
CIAPtrSEL = 2; (* printer SELECT *)
CIAPtrPOUT = 1; (* printer paper out *)
CIAPtrBUSY = 0; (* printer busy *)

(* ciab port B (0xbfd100) -- disk control *)
CIADskMOTOR = 7; (* disk motor *)
CIADskSEL3 = 6; (* disk select unit 3 *)
CIADskSEL2 = 5; (* disk select unit 2 *)
CIADskSEL1 = 4; (* disk select unit 1 *)
CIADskSELO = 3; (* disk select unit 0 *)
CIADskSIDE = 2; (* disk side select *)
CIADskDIREC = 1; (* disk direction of seek *)
CIADskSTEP = 0; (* disk step heads *)

VAR
(* ciaa is on an ODD address (e.g. the low byte) -- $bfe001 *)
(* ciab is on an EVEN address (e.g. the high byte) -- $bfd000 *)
(* initialized to point to the above specified hardware addresses *)
ciaa : CIAPtr;
ciab : CIAPtr;

END CIAHardware.

```


Module CIAResource

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: CIAResource.DEF      Version: Amiga.00.00      *
* Created: 11/29/86   Updated: 11/29/86   Author: Leon Frenkel      *
* Description: Amiga Complex Interface Adapter resource.      *
*****)
```

DEFINITION MODULE CIAResource;

FROM Interrupts IMPORT Interrupt, InterruptPtr;
FROM Resources IMPORT Resource;

CONST

CIAAName = "ciaa.resource";
CIABName = "ciab.resource";

PROCEDURE AbleICR(mask: BITSET; VAR resource: Resource): BITSET;
(* Enable/disable ICR interrupts.

mask - a bit mask indicating which interrupts to be modified
resource - a resource structure

result - the previous enable mask before the requested changes *)

PROCEDURE AddICRVector(iCRBit: CARDINAL; VAR interrupt: Interrupt;
VAR resource: Resource): InterruptPtr;
(* Attach an interrupt handler to a CIA bit.

iCRBit - bit number to set (0..4)
interrupt - interrupt structure
resource - resource structure

result - NIL if successful, otherwise ptr to current owner interrupt struc *)

PROCEDURE RemICRVector(iCRBit: CARDINAL; VAR interrupt: Interrupt;
VAR resource: Resource);
(* Detach an interrupt handler from a CIA bit.

iCRBit - bit number to set (0..4)
interrupt - interrupt structure
resource - resource structure *)

PROCEDURE SetICR(mask: BITSET; VAR resource: Resource): BITSET;
(* Cause, clear, and sample ICR interrupts.

mask - a bit mask indicating which interrupts to be effected
resource - resource structure

result - the previous interrupt register state before making the changes *)

END CIAResource.


```

(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: ClipboardDevice.DEF      Version: Amiga.00.00
* Created: 11/26/86   Updated: 11/26/86   Author: Leon Frenkel
* Description: "clipboard.device" types.
*****)

```

```
DEFINITION MODULE ClipboardDevice;
```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM IODevices IMPORT CmdNonStd, DevicePtr, IOFlagsSet, UnitPtr;
FROM Nodes IMPORT Node;
FROM Ports IMPORT Message;

```

```
CONST
```

```

  CBDPost      = CmdNonStd + 0;
  CBDCurrentReadID = CmdNonStd + 1;
  CBDCurrentWriteID = CmdNonStd + 2;

```

```
  CBErrObsoleteID = 1;
```

```
  PrimaryClip = 0; (* primary clip unit *)
```

```
TYPE
```

```

  ClipboardUnitPartialPtr = POINTER TO ClipboardUnitPartial;
  ClipboardUnitPartial = RECORD
    cuNode   : Node;      (* list of units *)
    cuUnitNum : LONGCARD; (* unit number for this unit *)
    (* the remaining unit data is private to the device *)
  END;

```

```

  IOClipReqPtr = POINTER TO IOClipReq;
  IOClipReq = RECORD

```

```

    ioMessage : Message;
    ioDevice  : DevicePtr; (* device node pointer *)
    ioUnit    : UnitPtr;   (* unit (driver private) *)
    ioCommand : CARDINAL;  (* device command *)
    ioFlags   : IOFlagsSet; (* including QUICK and SATISFY *)
    ioError   : BYTE;      (* error or warning num *)
    ioActual  : LONGCARD;  (* number of bytes transferred *)
    ioLength  : LONGCARD;  (* number of bytes requested *)
    ioData    : ADDRESS;   (* either clip stream or post port *)
    ioOffset  : LONGCARD;  (* offset in clip steam *)
    ioClipID  : LONGINT;   (* ordinal clip identifier *)
  END;

```

```

  SatisfyMsgPtr = POINTER TO SatisfyMsg;
  SatisfyMsg = RECORD

```

```

    smMsg      : Message; (* the length will be 6 *)
    smUnit     : CARDINAL; (* which clip unit this is *)
    smClipID   : LONGINT; (* the clip identifier of the post *)
  END;

```

```
END ClipboardDevice.
```

Module Clipping

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Clipping.DEF                      Version: Amiga.00.00
* Created: 11/24/86      Updated: 11/24/86      Author: Leon Frenkel
* Description: Clipping types including Layers.
*****)
```

DEFINITION MODULE Clipping;

```
FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM Graphics IMPORT BitMapPtr, Rectangle;
FROM Lists IMPORT MinList, List;
FROM Rasters IMPORT RastPortPtr;
FROM Regions IMPORT RegionPtr;
FROM Semaphores IMPORT SignalSemaphore;
```

TYPE

```
(* Pointer to types *)
LayerPtr      = POINTER TO Layer;
ClipRectPtr   = POINTER TO ClipRect;
LayerInfoPtr  = POINTER TO LayerInfo;
```

Layer = RECORD

```
  front      : LayerPtr; (* Ignored by roms *)
  back       : LayerPtr;
  ClipRect   : ClipRectPtr; (* read by roms to find first cliprect *)
  rp         : RastPortPtr; (* ignored by roms, I hope *)
  bounds     : Rectangle; (* ignored by roms *)
  reserved   : ARRAY [0..3] OF BYTE;
  priority   : CARDINAL; (* system use only *)
  Flags      : BITSET; (* obscured ?, Virtual BitMap? *)
  SuperBitMap : BitMapPtr;
```

```
(* super bitmap cliprects if VBitMap # NIL else damage cliprect
* list for refresh
*)
```

```
SuperClipRect : ClipRectPtr;
```

```
Window       : ADDRESS; (* reserved for user interface use *)
ScrollX      : INTEGER;
ScrollY      : INTEGER;
cr           : ClipRectPtr; (* used by dedice *)
cr2          : ClipRectPtr;
crnew        : ClipRectPtr;
SuperSaveClipRects : ClipRectPtr; (* preallocated cr's *)
cliprects    : ClipRectPtr; (* system use during refresh *)
LayerInfo    : LayerInfoPtr; (* points to head of the list *)
Lock         : SignalSemaphore;
reserved3    : ARRAY [0..7] OF BYTE;
ClipRegion   : RegionPtr;
saveClipRects : RegionPtr; (* used to back out when in trouble *)
reserved2    : ARRAY [0..21] OF BYTE;
DamageList   : RegionPtr; (* list of rectangles to refresh through *)
```

END;

ClipRect = RECORD

```
  next      : ClipRectPtr; (* roms used to find next ClipRect *)
  prev      : ClipRectPtr; (* ignored by roms, used by windowlib *)
  lobs      : LayerPtr; (* ignored by roms, used by windowlib *)
  BitMap     : BitMapPtr;
  bounds     : Rectangle; (* set up by windowlib, used by roms *)
  p1         : ClipRectPtr; (* system reserved *)
  p2         : ClipRectPtr; (* system reserved *)
  reserved   : LONGINT; (* system use *)
  Flags      : LONGINT; (* only exists in layer allocation *)
```

END;

```

LayerInfoFlags = (LayerSimple,
                  LayerSmart,
                  LayerSuper,
                  LIF3,
                  LayerUpdating,
                  LIF5,
                  LayerBackdrop,
                  LayerRefresh,

                  (* during BeginUpdate or during layerop *)
                  (* this happens if out of memory. *)
                  LayerClipRectsLost,

                  LIF9,
                  LIF10,
                  LIF11,
                  LIF12,
                  LIF13,
                  LIF14,
                  LIF15);

LayerInfoFlagsSet = SET OF LayerInfoFlags;

LayerInfo = RECORD
    toplayer      : LayerPtr;
    checklp       : LayerPtr; (* system use *)
    obs           : LayerPtr; (* system use *)
    FreeClipRects : MinList;
    Lock          : SignalSemaphore;
    gsHead        : List;      (* system use *)
    longreserved  : LONGINT;
    Flags         : LayerInfoFlagsSet;
    fattencount   : BYTE;
    LockLayersCount : BYTE;
    LayerInfoextrasize : CARDINAL;
    blitbuff      : ADDRESS;
    LayerInfoextra : ADDRESS;
END;

CONST
    (* internal cliprect flags *)
    CRNeedsNoConcealedRasters = 1;

    (* defines for code values for getcode *)
    IsLessX = 1;
    IsLessY = 2;
    IsGrtrX = 4;
    IsGrtrY = 8;

    NewLayerInfoCalled = 1;
    AlertLayersNoMem   = 83010000H;

END Clipping.

```

Module CList

```
(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: CList.DEF               Version: Amiga.00.00
* Created: 11/28/86   Updated: 11/28/86   Author: Leon Frenkel
* Description: "clist.library" functions.
* WARNING: Version 1.2 of the ROM Kernel does not have the "clist.library"
*          included. So an OpenLibrary() will fail. However the library
*          may be restored in future versions of the ROM Kernel, in which
*          case this module may again become useful again.
*****)
```

DEFINITION MODULE CList;

FROM SYSTEM IMPORT ADDRESS, BYTE, WORD;
FROM Libraries IMPORT LibraryPtr;

CONST

CListName = "clist.library";

VAR

CListBase : LibraryPtr;

TYPE

CListPool = ADDRESS;

CListDesc = LONGINT;

PROCEDURE AllocCList(cLPool: CListPool): CListDesc;

(* Allocate and initialize a clist.

 cLPool - a clist pool that has already been initialized

 result - a longword descriptor for a clist that can be used for clist func
 if result < 0 then no space available for new clist *)

PROCEDURE ConcatCList(sourceCList, destCList: CListDesc): LONGCARD;

(* Concatenate two character lists.

 sourceCList - source clist

 destCList - destination clist

 error - if result # 0 then ran out of memory *)

PROCEDURE CopyCList(cList: CListDesc): CListDesc;

(* Copy a clist to a new clist.

 cList - original clist

 result - new copy of clist, if result < 0 then not enough memory *)

PROCEDURE FlushCList(cList: CListDesc);

(* Clear a character list.

 cList - clist to be flushed *)

PROCEDURE FreeCList(cList: CListDesc);

(* Free a clist.

 cList - clist to be freed *)

PROCEDURE GetCLBuf(cList: CListDesc; buffer: ADDRESS;

 maxLength: LONGCARD): LONGCARD;

(* Convert a character list to contiguous data.

 cList - clist descriptor

 buffer - pointer to buffer to be used for operation

maxLength - the maximum size of buffer

result - number of bytes actually copied into buffer. Never greater than
maxLength. The remaining characters are left in the clist *)

PROCEDURE GetCLChar(cList: CLISTDesc): LONGINT;
(* Get a byte from the beginning of a character list.

cList - clist from which to get byte

result - the byte from the beginning of the clist or -1 if no data *)

PROCEDURE GetCLWord(cList: CLISTDesc): LONGINT;
(* Get a word from the beginning of a character list.

cList - clist from which to get word

result - the word from the beginning of the clist or -1 if no data *)

PROCEDURE IncrCLMark(cList: CLISTDesc): LONGCARD;
(* Increment a clist mark to the next position.

cList - clist in which to increment mark

result - non-zero if the next offset is not in the clist *)

PROCEDURE InitCLPool(cLPool: CLISTPool; size: LONGCARD): LONGCARD;
(* Initialize a clist pool.

cLPool - data area to be used as clist pool
size - size of the pool, in bytes. Maximum of 16M bytes.

result - non-zero if pool size is too small to setup clist management *)

PROCEDURE MarkCLIST(cList: CLISTDesc; offset: LONGCARD): LONGCARD;
(* Mark a position in a clist.

cList - clist to set the mark in
offset - byte offset into clist. The first byte is at offset zero.

result - non-zero if the offset is not in the clist *)

PROCEDURE PeekCLMark(cList: CLISTDesc): CHAR;
(* Peek at the byte in the clist at the mark.

cList - clist from which to peek

result - the byte(char) at the mark in the clist *)

PROCEDURE PutCLBuf(cList: CLISTDesc; buffer: ADDRESS; length: LONGCARD): LONGCARD;
(* Convert contiguous data into a character list.

cList - clist into which to move the data
buffer - pointer to the buffer from which to read the data
length - number of bytes of data in the buffer

result - non-zero indicates the number of bytes not added *)

PROCEDURE PutCLChar(cList: CLISTDesc; byte: BYTE): LONGCARD;
(* Add a byte to the end of a clist.

cList - clist to be used for operation
byte - the byte to add to end of clist. (Any object which occupies 8-bits)

result - non-zero indicate the byte could not be added *)

Module CList

PROCEDURE PutCLWord(cList: CListDesc; word: WORD): LONGCARD;
(* Add a word to the end of a clist.

cList - clist to be used for operation
word - the word to add to end of clist. (Any object which occupies 16-bits)

result - non-zero indicates number the number of bytes not added. Partial
words are not added, so result is always zero or two *)

PROCEDURE SizeCList(cList: CListDesc): LONGCARD;
(* Get the number of bytes in a clist.

cList - clist for which to determine size

result - the number of bytes in the clist *)

PROCEDURE SplitCList(cList: CListDesc): CListDesc;
(* Split a clist.

cList - clist to be split

result - new clist that contains the tail end of the original clist. if
not enough memory to build new clist, or the mark is invalid
result is negative *)

PROCEDURE SubCList(cList: CListDesc; index, length: LONGCARD): CListDesc;
(* Copy a substring from a clist.

cList - clist to be used for operation
index - offset into the clist to start copying the substring from
length - number of bytes to copy

result - clist containing the substring, if result < 0 then not enough
space available for the new clist *)

PROCEDURE UnGetCLChar(cList: CListDesc; byte: BYTE): LONGCARD;
(* Add a byte to the beginning of a clist.

cList - clist to be used for operation
byte - byte to add to beginning of clist

result - non-zero indicates the byte could not be added *)

PROCEDURE UnGetCLWord(cList: CListDesc; word: WORD): LONGCARD;
(* Add a word to the beginning of a clist.

cList - clist to be used for operation
word - word to add to beginning of clist

result - non-zero indicates number the number of bytes not added. Partial
words are not added, so result is always zero or two *)

PROCEDURE UnPutCLChar(cList: CListDesc): LONGINT;
(* Get a byte from the end of a clist.

cList - clist from which to get the byte

result - the byte from the end of the clist. if no data available then -1 *)

PROCEDURE UnPutCLWord(cList: CListDesc): LONGINT;
(* Get a word from the end of a clist.

cList - clist from which to get the word

result - the word from the end of the clist or -1 if not enough data *)

END CList.


```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1987 by Leon Frenkel                *
*                               *
*****
* Name: ConfigRegs.DEF      Version: Amiga.00.00      *
* Created: 07/22/87   Updated: 07/22/87   Author: Leon Frenkel      *
* Description: Register and bit definitions for expansion boards.      *
*****)

```

DEFINITION MODULE ConfigRegs;

FROM SYSTEM IMPORT BYTE;

```

(*)
* many of the constants below consist of a triplet of equivalent
* definitions: xxMASK is a bit mask of those bits that matter.
* xxBIT is the starting bit number of the field.  xxSIZE is the
* number of bits that make up the definition.  This method is
* used when the field is larger than one bit.
*
* If the field is only one bit wide then the xxB_xx and xxF_xx convention
* is used (xxB_xx is the bit number, and xxF_xx is mask of the bit).
*)

```

CONST

(* manifest constants *)

ESlotSize = 10000H;

ESlotMask = 0FFFFH;

ESlotShift = 16;

(* these define the two free regions of Zorro memory space.

* THESE MAY WELL CHANGE FOR FUTURE PRODUCTS!

*)

EExpansionBase = 0E80000H;

EExpansionSize = 0080000H;

EExpansionSlots = 8;

EMemoryBase = 200000H;

EMemorySize = 800000H;

EMemorySlots = 128;

(* erType definitions *)

(* board type -- ignore "old style" boards *)

ERTTypeMask = BYTE(0COH);

ERTTypeBit = BYTE(6);

ERTTypeSize = BYTE(2);

ERTNewBoard = BYTE(0COH);

(* type field memory size *)

ERTMemMask = BYTE(7);

ERTMemBit = BYTE(0);

ERTMemSize = BYTE(3);

(* other bits defined in type field *)

ERTChainedConfig = 3;

ERTDiagValid = 4;

ERTMemList = 5;

(* erFlags byte -- for those things that didn't fit into the type byte *)

ERFMemSpace = 7; (* wants to be in 8 meg space. Also

* implies that board is moveable

*)

ERFNoShutup = 6; (* board can't be shut up. Must not

* be a board. Must be a box that

* does not pass on the bus.

*)

(* figure out amount of memory needed by this box/board

* #define ERT_MEMNEEDED(t)

((t)&ERT_MEMMASK)? 0x10000 << (((t)&ERT_MEMMASK) - 1) : 0x800000)

* same as ERT_MEMNEEDED, but return number of slots

* #define ERT_SLOTSNEEDED(t)

((t)&ERT_MEMMASK)? 1 << (((t)&ERT_MEMMASK) - 1) : 0x80)

Module ConfigRegs

```

*)

(* interrupt control register *)
ECIIntEna      = 1;
ECIReset       = 3;
ECIInt2Pend    = 4;
ECIInt6Pend    = 5;
ECIInt7Pend    = 6;
ECIInterrupting = 7;

(*)

* convert a expansion slot number into a memory address
* #define EC_MEMADDR(slot)          ((slot) << (E_SLOTSHIFT) )
*
* a kludge to get the byte offset of a structure
* #define EROFFSET(er)              ((int)&((struct ExpansionRom * )0)->er)
* #define ECOFFSET(ec)              \
* (sizeof(struct ExpansionRom)+((int)&((struct ExpansionControl * )0)->ec))
*)

(*)
* Expansion boards are actually organized such that only one nibble per
* word (16 bits) are valid information. This table is structured
* as LOGICAL information. This means that it never corresponds
* exactly with a physical implementation.
*
* The expansion space is logically split into two regions:
* a rom portion and a control portion. The rom portion is
* actually stored in one's complement form (except for the
* er_type field).
*)

TYPE
ExpansionRomPtr = POINTER TO ExpansionRom;
ExpansionRom = RECORD
    erType      : BYTE;
    erProduct   : BYTE;
    erFlags     : BYTE;
    erReserved03 : BYTE;
    erManufacturer : CARDINAL;
    erSerialNumber : LONGCARD;
    erInitDiagVec : CARDINAL;
    erReserved0c : BYTE;
    erReserved0d : BYTE;
    erReserved0e : BYTE;
    erReserved0f : BYTE;
END;

ExpansionControlPtr = POINTER TO ExpansionControl;
ExpansionControl = RECORD
    ecInterrupt : BYTE; (* interrupt control register *)
    ecReserved11 : BYTE;
    ecBaseAddress : BYTE; (* set new config address *)
    ecShutup      : BYTE; (* don't respond, pass config out*)
    ecReserved14 : BYTE;
    ecReserved15 : BYTE;
    ecReserved16 : BYTE;
    ecReserved17 : BYTE;
    ecReserved18 : BYTE;
    ecReserved19 : BYTE;
    ecReserved1a : BYTE;
    ecReserved1b : BYTE;
    ecReserved1c : BYTE;
    ecReserved1d : BYTE;
    ecReserved1e : BYTE;
    ecReserved1f : BYTE;
END;

(*****
*
* these are the specifications for the diagnostic area. If the Diagnostic
* Address Valid bit is set in the Board Type byte (the first byte in
* expansion space) then the Diag Init vector contains a valid offset.
*
* The Diag Init vector is actually a word offset from the base of the

```



```

* board. The resulting address points to the base of the DiagArea
* structure. The structure may be physically implemented either four,
* eight, or sixteen bits wide. The code will be copied out into
* ram first before being called.
*
* The da_Size field, and both code offsets (da_DiagPoint and da_BootPoint)
* are offsets from the diag area AFTER it has been copied into ram, and
* "de-nibbleized" (if needed). In other words, the size is the size of
* the actual information, not how much address space is required to
* store it.
*
* All bits are encoded with uninverted logic (e.g. 5 volts on the bus
* is a logic one).
*
* If your board is to make use of the boot facility then it must leave
* its config area available even after it has been configured. Your
* boot vector will be called AFTER your board's final address has been
* set.
*
*****
CONST
(* daConfig definitions *)
DACBusWidth = BYTE(0COH); (* two bits for bus width *)
DACNibbleWide = BYTE(000H);
DACByteWide = BYTE(040H);
DACWordWide = BYTE(080H);

DACBootTime = BYTE(030H); (* two bits for when to boot *)
DACNever = BYTE(000H); (* obvious *)
DACConfigTime = BYTE(010H); (* call daBootPoint when first config the device*)
DACBindTime = BYTE(020H); (* run when binding drivers to board *)

```

```

TYPE
DiagAreaPtr = POINTER TO DiagArea;
DiagArea = RECORD
    daConfig : BYTE; (* see above for definitions *)
    daFlags : BYTE; (* see above for definitions *)
    daSize : CARDINAL; (* size (in bytes) of total diag area *)
    daDiagPoint : CARDINAL; (* start for diagnostics, or zero *)
    daBootPoint : CARDINAL; (* start for booting *)
    daName : CARDINAL; (* offset in diag area where a string *)
    (* identifier can be found (or zero *)
    (* if no identifier is present). *)
    daReserved01 : CARDINAL; (* two words of reserved data. must *)
    daReserved02 : CARDINAL; (* be zero. *)
END;

```

```

(*)
* These are the calling conventions for Diag or Boot area
*
* A7 -- points to at least 2K of stack
* A6 -- ExecBase
* A5 -- ExpansionBase
* A3 -- your board's ConfigDev structure
* A2 -- Base of diag/init area that was copied
* A0 -- Base of your board
*
* Your board should return a value in D0. If this value is NULL, then
* the diag/init area that was copied in will be returned to the free
* memory pool.
*)

```

```
END ConfigRegs.
```

Module ConsoleDevice

```
(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: ConsoleDevice.DEF      Version: Amiga.00.00
* Created: 11/26/86   Updated: 05/07/87   Author: Leon Frenkel
* Description: "console.device" types.
*****)
```

DEFINITION MODULE ConsoleDevice;

FROM SYSTEM IMPORT ADDRESS;
FROM KeyMapResource IMPORT KeyMapPtr;
FROM InputEvents IMPORT InputEvent;
FROM IODevices IMPORT CmdNonStd, DevicePtr;

CONST
 ConsoleName = "console.device";

VAR
 ConsoleBase : DevicePtr;

CONST

(* Console commands *)
 CDAskKeyMap = CmdNonStd + 0;
 CDSetKeyMap = CmdNonStd + 1;
 CDAskDefaultKeyMap = CmdNonStd + 2;
 CDSetDefaultKeyMap = CmdNonStd + 3;

(* SGR parameters *)
 SGRPrimary = 0;
 SGRBold = 1;
 SGRItalic = 3;
 SGRUnderscore = 4;
 SGRNegative = 7;

(* these names refer to the ANSI standard, not the implementation *)
 SGRBlack = 30;
 SGRRed = 31;
 SGRGreen = 32;
 SGRYellow = 33;
 SGRBlue = 34;
 SGRMagenta = 35;
 SGRCyan = 36;
 SGRWhite = 37;
 SGRDefault = 39;

SGRBlackBG = 40;
 SGRRedBG = 41;
 SGRGreenBG = 42;
 SGRYellowBG = 43;
 SGRBlueBG = 44;
 SGRMagentaBG = 45;
 SGRCyanBG = 46;
 SGRWhiteBG = 47;
 SGRDefaultBG = 49;

```

(* these names refer to the implementation, they are the preferred *)
(* names for use with the Amiga console device. *)
SGRC1r0 = 30;
SGRC1r1 = 31;
SGRC1r2 = 32;
SGRC1r3 = 33;
SGRC1r4 = 34;
SGRC1r5 = 35;
SGRC1r6 = 36;
SGRC1r7 = 37;

SGRC1r0BG = 40;
SGRC1r1BG = 41;
SGRC1r2BG = 42;
SGRC1r3BG = 43;
SGRC1r4BG = 44;
SGRC1r5BG = 45;
SGRC1r6BG = 46;
SGRC1r7BG = 47;

(* DSR parameters *)
DSRCPR = 6;

(* CTC parameters *)
CTCHSetTab = 0;
CTCHC1rTab = 2;
CTCHC1rTabsAll = 5;

(* TBC parameters *)
TBCHC1rTab = 0;
TBCHC1rTabsAll = 3;

(* SM and RM parameters *)
MLNM = 20; (* linefeed newline mode *)
MASM = ">1"; (* auto scroll mode *)
MAWM = "?7"; (* auto wrap mode *)

PROCEDURE RawKeyConvert(VAR event: InputEvent; buffer: ADDRESS;
    length: LONGCARD; keyMap: KeyMapPtr): LONGINT;
(* Decode raw input classes.

    event - InputEvent structure
    buffer - a byte buffer large enough to hold all anticipated chars generated
            by this conversion
    length - maximum anticipation, i.e. the buffer size in bytes
    keyMap - a KeyMap structure pointer, or NIL if the default console device
            keymap is to be used

    result - number of chars in buffer or -1 if buffer overflow occurred *)
END ConsoleDevice.

```

Module ConsoleDevUnit

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: ConsoleDevUnit.DEF      Version: Amiga.00.00
* Created: 11/26/86   Updated: 05/21/87   Author: Leon Frenkel
* Description: Console device unit definitions.
*****)

DEFINITION MODULE ConsoleDevUnit;

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM ConsoleDevice IMPORT MLNM;
FROM InputEvents IMPORT IECClassMax;
FROM KeyMapResource IMPORT KeyMap;
FROM Ports IMPORT MsgPort;

CONST
  PMBASM = MLNM + 1; (* internal storage bit for AS flag *)
  PMBAWM = PMBASM + 1; (* internal storage bit for AW flag *)
  MaxTabs = 80;

TYPE
  ConUnitPtr = POINTER TO ConUnit;
  ConUnit = RECORD
    cuMP      : MsgPort;
    (* read only variables *)
    cuWindow  : ADDRESS; (* WindowPtr intui window bound to this unit *)
    cuXCP     : INTEGER; (* character position *)
    cuYCP     : INTEGER;
    cuXMax    : INTEGER; (* max character position *)
    cuYMax    : INTEGER;
    cuXRSize  : INTEGER; (* character raster size *)
    cuYRSize  : INTEGER;
    cuXROrigin : INTEGER; (* raster origin *)
    cuYROrigin : INTEGER;
    cuXRExtant : INTEGER; (* raster maxima *)
    cuYRExtant : INTEGER;
    cuXMinShrink : INTEGER; (* smallest area intact from resize process *)
    cuYMinShrink : INTEGER;
    cuXCCP    : INTEGER; (* cursor position *)
    cuYCCP    : INTEGER;

    (* ---- read/write variables (writes must be protected) *)
    (* ---- storage for AskKeyMap and SetKeyMap *)
    cuKeyMapStruct : KeyMap;
    (* tab stops, 0 at start, $FFFF at end of list *)
    cuTabStops    : ARRAY [0..MaxTabs-1] OF CARDINAL;

    (* console rastport attributes *)
    cuMask      : BYTE;
    cuFgPen     : BYTE;
    cuBgPen     : BYTE;
    cuAOLPen    : BYTE;
    cuDrawMode  : BYTE; (* DrawModeSet *)
    cuAreaPtSz  : BYTE;
    cuAreaPtrn  : ADDRESS; (* cursor area pattern *)
    cuMinTerms  : ARRAY [0..7] OF BYTE; (* console minterms *)
    cuFont      : ADDRESS; (* TextFontPtr *)
    cuAlgoStyle : BYTE;
    cuTxFlags   : BYTE;
    cuTxHeight  : CARDINAL;
    cuTxWidth   : CARDINAL;
    cuTxBaseline : CARDINAL;
    cuTxSpacing : CARDINAL;

    (* console MODES and RAW EVENTS switches *)
    cuModes     : ARRAY [0..((PMBASM+7) DIV 8)-1] OF BYTE;
    cuRawEvents : ARRAY [0..((CARDINAL(IECClassMax)+7) DIV 8)-1] OF BYTE;
  END;

END ConsoleDevUnit.

```

```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: Conversions.DEF          Version: Amiga.00.00
* Created: 01/17/87   Updated: 07/31/87   Author: Leon Frenkel
* Description: Functions for converting a number to a string and a string
*              to a number.
*****)

```

```

DEFINITION MODULE Conversions;

```

```

FROM SYSTEM IMPORT LONGWORD;

```

```

(*$L+*)

```

```

PROCEDURE ConvStringToNumber(string: ARRAY OF CHAR; VAR num: LONGWORD;
                             signed: BOOLEAN; base: CARDINAL): BOOLEAN;

```

```

(* Given a string which is supposed to contain an char representation of
   a number convert it to a binary representation of the number. The base of
   the number is specified with the parm "base": for example 10 = decimal,
   16 = hex, 2 = binary. The string passed to the function may contain
   leading spaces which will be ignored. If the parm "signed" is TRUE then
   a "+" or "-" will be accepted and handled appropriately. If the conversion
   is successful the function returns TRUE and parm "num" is valid. Otherwise
   returns FALSE, and "num" is undefined. The string must terminate on a
   character <= " " to be considered valid.

```

```

   string - string to parsed for a number
   num - variable to store the binary number into (any 32-bit type)
   signed - if TRUE then a signed number is valid, FALSE if unsigned
   base - the base of the number to be parsed

```

```

   result - TRUE if successful conversion, otherwise FALSE *)

```

```

PROCEDURE ConvNumberToString(VAR string: ARRAY OF CHAR; num: LONGWORD;
                              signed: BOOLEAN; base: CARDINAL;
                              digits: CARDINAL; padChar: CHAR);

```

```

(* Given a binary representation of a number convert it into a string
   representation. If the string was not large enough to accomodate the
   number, only the part that fit will be stored in the string.

```

```

   string - string large enough to accomodate the number and a null-terminator.
   num - number to be converted (any 32-bit type)
   signed - TRUE = high bit represents sign of number, FALSE = unsigned number
   base - base of number such as 2=binary, 16=hex, 10=decimal, etc...
   digits - the generating string should have # of digits
   padChar - char to use for leading 0's, usually "0" or " " *)

```

```

END Conversions.

```

Module Copper

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Copper.DEF                      Version: Amiga.00.00
* Created: 11/20/86      Updated: 05/08/87      Author: Leon Frenkel
* Description: Graphics Copper types/functions.
*****)

```

DEFINITION MODULE Copper;

FROM SYSTEM IMPORT ADDRESS, WORD;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)

CONST

```

CopperMove = 0;      (* pseudo opcode for move #XXXX,dir *)
CopperWait = 1;      (* pseudo opcode for wait y,x *)
CprNxtBuf = 2;      (* continue processing with next buffer *)
CprNtLof = 8000H;    (* copper instruction only for short frames *)
CprNtSht = 4000H;    (* copper instruction only for long frames *)

```

TYPE

CopListPtr = POINTER TO CopList;

CopInsPtr = POINTER TO CopIns;

CopIns = RECORD

CASE OpCode : INTEGER OF

```

| CprNxtBuf : NxtList : CopListPtr;
| CopperWait: VWaitPos : INTEGER; (* vertical beam wait *)
              HWaitPos : INTEGER; (* horizontal beam wait *)
| CopperMove: DestAddr : INTEGER; (* destination adr of move*)
              DestData : INTEGER; (* immediate data to send *)

```

END;

END;

(* structure of cprlist that points to list that hardware actually executes *)

cprlistPtr = POINTER TO cprlist;

cprlist = RECORD

```

Next      : cprlistPtr;
start     : ADDRESS;    (* start of copper list *)
MaxCount  : INTEGER;    (* number of long instructions *)

```

END;

CopList = RECORD

```

Next      : CopListPtr; (* next block for this copper list *)
CopList   : CopListPtr; (* system use *)
ViewPort  : ADDRESS;    (* system use ViewPortPtr, circular ref *)
CopIns    : CopInsPtr;  (* start of this block *)
CopPtr    : CopInsPtr;  (* intermidate ptr *)
CopLStart : ADDRESS;    (* mrgcop fills this in for Long Frame *)
CopsStart : ADDRESS;    (* mrgcop fills this in for Short Frame *)
Count     : INTEGER;    (* intermediate counter *)
MaxCount  : INTEGER;    (* max # of copins for this block *)
DyOffset  : INTEGER;    (* offset this copper list vertical waits*)

```

END;

UCopListPtr = POINTER TO UCopList;

UCopList = RECORD

```

Next      : UCopListPtr;
FirstCopList : CopListPtr; (* head node of this copper list *)
CopList    : CopListPtr; (* node in use *)

```

END;

copinitPtr = POINTER TO copinit;

copinit = RECORD

```

diagstr : ARRAY [0..3] OF CARDINAL; (* copperlist for first bitplane *)
sprstrup : ARRAY [0..((2*8*2)+2+(2*2)+2)-1] OF CARDINAL;
sprstop  : ARRAY [0..1] OF CARDINAL;

```

END;

PROCEDURE CBump(VAR c: UCopList);

(* Increment user copper list pointer (bump to next position in list).

c - UCopList structure *)

```
PROCEDURE CMove(VAR c: UCopList; a: ADDRESS; v: WORD);
(* Append copper move instruction to user copper list.
```

```
  c - UCopList structure
  a - hardware register
  v - 16 bit value to be written *)
```

```
PROCEDURE CWait(VAR c: UCopList; v, h: INTEGER);
(* Append copper wait instruction to user copper list.
```

```
  c - UCopList structure
  v - vertical beam position (relative to top of viewport)
  h - horizontal beam position *)
```

```
PROCEDURE FreeCopList(VAR coplist: CopList);
(* Deallocate intermediate copper list.
```

```
  coplist - CopList structure *)
```

```
PROCEDURE FreeCprList(VAR cprlist: cprlist);
(* Deallocate hardware copper list.
```

```
  cprlist - cprlist structure *)
```

```
PROCEDURE UCopperListInit(VAR c: UCopList; n: INTEGER): UCopListPtr;
(* Initialize user copperlist to accept intermediate user copper instructions.
```

```
  c - UCopList structure
  n - number of instruction buffer must hold
```

```
  result - an initialized list to accept intermediate copper instructions *)
```

```
END Copper.
```


Module CopperUtil

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: CopperUtil.DEF              Version: Amiga.00.00      *
* Created: 01/21/87   Updated: 01/21/87   Author: Leon Frenkel *
* Description: Graphics Copper Utility Functions. Implimentations of "C" *
* macros. *
*****)

DEFINITION MODULE CopperUtil;

FROM SYSTEM IMPORT ADDRESS, WORD;
FROM Copper IMPORT UCopList, UCopListPtr;

(*$L+*)

PROCEDURE CEND(VAR c: UCopList);
(* Terminate user copper list.
* 'C' macro: CEND(c) { CWAIT(c,10000,255); }

c - UCopList structure *)

PROCEDURE CINIT(VAR c: UCopList; n: CARDINAL): UCopListPtr;
(* Initialize user copperlist to accept intermediate user copper instructions.
* 'C' macro: CINIT(c,n) { UCopperListInit(c,n); }

c - UCopList structure
n - number of instructions buffer must hold.

result - an initialized list to accept intermediate copper instructions *)

PROCEDURE CMOVE(VAR c: UCopList; a: ADDRESS; v: WORD);
(* Add instruction to move value v to hardware register a.
* 'C' macro: CMOVE(c,a,b) { CMove(c,&a,b); CBump(c); }

c - UCopList structure
a - hardware register
v - 16 bit value to be written *)

PROCEDURE CWAIT(VAR c: UCopList; v, h: INTEGER);
(* Append copper wait instruction to user copper list.
* 'C' macro: CWAIT(c,a,b) { CWait(c,a,b); CBump(c); }

c - UCopList structure
v - vertical beam position (relative to top of viewport)
h - horizontal beam position *)

END CopperUtil.

```



```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: CustomHardware.DEF      Version: Amiga.00.00      *
* Created: 11/29/86   Updated: 01/15/87   Author: Leon Frenkel      *
* Description: Amiga Custom Hardware definitions.      *
*****)

```

```
DEFINITION MODULE CustomHardware;
```

```
FROM SYSTEM IMPORT ADDRESS;
```

```
(*$L+*)
```

```
TYPE
```

```

  AudChannelPtr = POINTER TO AudChannel;
  AudChannel = RECORD
    acptr : ADDRESS; (* ptr to start of waveform data *)
    aclen : CARDINAL; (* length of waveform in words *)
    acper : CARDINAL; (* sample period *)
    acvol : CARDINAL; (* volume *)
    acdat : CARDINAL; (* sample pair *)
    acpad : ARRAY [0..1] OF CARDINAL; (* unused *)
  END;

```

```

  SpriteDefPtr = POINTER TO SpriteDef;
  SpriteDef = RECORD
    pos : CARDINAL;
    ctl : CARDINAL;
    dataa : CARDINAL;
    datab : CARDINAL;
  END;

```

```

  CustomPtr = POINTER TO Custom;
  Custom = RECORD

```

```

    bltddat : CARDINAL;
    dmaconr : CARDINAL;
    vposr : CARDINAL;
    vhposr : CARDINAL;
    dskdatr : CARDINAL;
    joyOdat : CARDINAL;
    joyldat : CARDINAL;
    clxdat : CARDINAL;
    adkconr : CARDINAL;
    potOdat : CARDINAL;
    potldat : CARDINAL;
    potinp : CARDINAL;
    serdatr : CARDINAL;
    dskbytr : CARDINAL;
    intenar : CARDINAL;
    intreqr : CARDINAL;
    dskpt : ADDRESS;
    dsklen : CARDINAL;
    dskdat : CARDINAL;
    refptr : CARDINAL;
    vposw : CARDINAL;
    vhposw : CARDINAL;
    copcon : CARDINAL;
    serdat : CARDINAL;
    serper : CARDINAL;
    potgo : CARDINAL;
    joytest : CARDINAL;
    strequ : CARDINAL;
    strvbl : CARDINAL;
    strhor : CARDINAL;
    strlong : CARDINAL;
    bltcon0 : BITSET;
    bltcon1 : BITSET;
    bltafwm : CARDINAL;
    bltalwm : CARDINAL;
    bltcpt : ADDRESS;
    bltbpt : ADDRESS;
    bltapt : ADDRESS;
    bltdpt : ADDRESS;

```

Module CustomHardware

```

bltsize : CARDINAL;
pad2d : ARRAY [0..2] OF CARDINAL;
bltcmmod : CARDINAL;
bltbmod : CARDINAL;
bltamod : CARDINAL;
bltdmod : CARDINAL;
pad34 : ARRAY [0..3] OF CARDINAL;
bltcdat : CARDINAL;
bltbdat : CARDINAL;
bltatdat : CARDINAL;
pad3b : ARRAY [0..3] OF CARDINAL;
dsksync : CARDINAL;
cop1lc : LONGCARD;
cop2lc : LONGCARD;
copjmp1 : CARDINAL;
copjmp2 : CARDINAL;
copins : CARDINAL;
diwstrt : CARDINAL;
diwstop : CARDINAL;
ddfstrt : CARDINAL;
ddfstop : CARDINAL;
dmacon : BITSET;
clxcon : CARDINAL;
intena : BITSET;
intreq : CARDINAL;
adkcon : CARDINAL;
aud : ARRAY [0..3] OF AudChannel;
bplpt : ARRAY [0..5] OF ADDRESS;
pad7c : ARRAY [0..3] OF CARDINAL;
bplcon0 : CARDINAL;
bplcon1 : CARDINAL;
bplcon2 : CARDINAL;
pad83 : CARDINAL;
bpl1mod : CARDINAL;
bpl2mod : CARDINAL;
pad86 : ARRAY [0..1] OF CARDINAL;
bpldat : ARRAY [0..5] OF CARDINAL;
pad8e : ARRAY [0..1] OF CARDINAL;
sprpt : ARRAY [0..7] OF ADDRESS;
spr : ARRAY [0..7] OF SpriteDef;
color : ARRAY [0..31] OF CARDINAL;
END;
```

VAR

```

(* Ptr initialized to point to address of hardware: 00DFF000H *)
custom : CustomPtr;
```

END CustomHardware.

```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: DiskFont.DEF                               Version: Amiga.00.00
* Created: 11/28/86   Updated: 11/28/86   Author: Leon Frenkel
* Description: "diskfont.library" types/functions.
*****)

DEFINITION MODULE DiskFont;

FROM SYSTEM IMPORT ADDRESS;
FROM Libraries IMPORT LibraryPtr;
FROM Nodes IMPORT Node;
FROM Text IMPORT FontFlagsSet, FontStyleSet, TextAttr, TextFont, TextFontPtr;

CONST
  DiskFontName = "diskfont.library";

VAR
  DiskFontBase : LibraryPtr;

CONST
  MaxFontPath = 256; (* including null terminator *)

TYPE
  FontContentsPtr = POINTER TO FontContents;
  FontContents = RECORD
    fcFileName : ARRAY [0..MaxFontPath-1] OF CHAR;
    fcYSize    : CARDINAL;
    fcStyle    : FontStyleSet;
    fcFlags    : FontFlagsSet;
  END;

CONST
  FCHID = OF00H;

TYPE
  FontContentsHeaderPtr = POINTER TO FontContentsHeader;
  FontContentsHeader = RECORD
    fchFileID : CARDINAL; (* FCHID *)
    fchNumEntries : CARDINAL;
    (* fchFC : ARRAY [0..fchNumEntries-1] OF FontContents*)
  END;

CONST
  DFHID = OF80H;
  MaxFontName = 32; (* font name including ".font\0" *)

TYPE
  (* the following 8 bytes are not actually considered a part of the *)
  (* DiskFontHeader, but immediately precede it. The NextSegment is *)
  (* supplied by the linker/loader, and the ReturnCode is the code *)
  (* at the beginning of the font in case someone runs it... *)
  (* ULONG dfh_NextSegment; actually a BPTR *)
  (* ULONG dfh_ReturnCode; MOVEQ #0,DO : RTS *)
  (* here then is the official start of the DiskFontHeader... *)
  DiskFontHeaderPtr = POINTER TO DiskFontHeader;
  DiskFontHeader = RECORD
    dfhDF : Node; (* node to link disk fonts *)
    dfhFileID : CARDINAL; (* DFHID *)
    dfhRevision : CARDINAL; (* font revision *)
    dfhSegment : LONGINT; (* segment address when loaded *)
    dfhName : ARRAY [0..MaxFontName-1] OF CHAR; (* name *)
    dfhTF : TextFont; (* loaded TextFont structure *)
  END;

  AvailFontType = (AFMemory,
    AFDisk,
    AF2, AF3, AF4, AF5, AF6, AF7, AF8,
    AF9, AF10, AF11, AF12, AF13, AF14, AF15);
  AvailFontTypeSet = SET OF AvailFontType;

```

Module DiskFont

```
AvailFontPtr = POINTER TO AvailFont;
AvailFont = RECORD
    afType : AvailFontTypeSet; (* Memory or Disk *)
    afAttr : TextAttr;          (* text attributes for font *)
END;

AvailFontsHeaderPtr = POINTER TO AvailFontsHeader;
AvailFontsHeader = RECORD
    afhNumEntries : CARDINAL; (* # of AvailFonts elements *)
    (* afhAF : ARRAY [0..afhNumEntries-1] OF AvailFont *)
END;

PROCEDURE AvailFonts(buffer: ADDRESS; bufBytes: LONGCARD;
    types: AvailFontTypeSet): LONGCARD;
(* Build an array of all fonts in memory / on disk.

    buffer - memory to be filled with struct AvailFontsHeader
    bufBytes - number of bytes in buffer
    types - Specify memory or disk fonts or both.

    result - if non-zero indicates number of bytes needed in addition to those
    supplied to create structure *)

PROCEDURE OpenDiskFont(VAR textAttr: TextAttr): TextFontPtr;
(* Load and get a pointer to a disk font.

    textAttr - a TextAttr structure that describes the text font attributes

    result - a pointer to a TextFont structure *)

END DiskFont.
```

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: DiskResource.DEF                      Version: Amiga.00.00      *
* Created: 11/29/86   Updated: 05/07/87   Author: Leon Frenkel        *
* Description: Amiga Disk resource.                                         *
*****)

```

```

DEFINITION MODULE DiskResource;

```

```

FROM SYSTEM IMPORT BYTE;
FROM Interrupts IMPORT Interrupt;
FROM Libraries IMPORT Library, LibraryPtr, LibBase, LibVectSize;
FROM Lists IMPORT List;
FROM Ports IMPORT Message;

```

```

CONST

```

```

    DiskName = "disk.resource";

```

```

TYPE

```

```

    (* Resource structures *)

```

```

    DiscResourceUnitPtr = POINTER TO DiscResourceUnit;

```

```

    DiscResourceUnit = RECORD

```

```

        druMessage      : Message;
        druDiscBlock    : Interrupt;
        druDiscSync     : Interrupt;
        druIndex        : Interrupt;

```

```

    END;

```

```

    (* drFlags entries *)

```

```

    DiscResourceFlags = (DRA1loc0, (* unit zero is allocated *)
                        DRA1loc1, (* unit one is allocated *)
                        DRA1loc2, (* unit two is allocated *)
                        DRA1loc3, (* unit three is allocated *)
                        DRF4,
                        DRF5,
                        DRF6,
                        DRAActive); (* is the disk currently busy? *)

```

```

    DiscResourceFlagsSet = SET OF DiscResourceFlags;

```

```

    DiscResourcePtr = POINTER TO DiscResource;

```

```

    DiscResource = RECORD

```

```

        drLibrary      : Library;
        drCurrent      : DiscResourceUnitPtr;
        drFlags        : DiscResourceFlagsSet;
        drpad          : BYTE;
        drSysLib       : LibraryPtr;
        drCiaResource  : LibraryPtr;
        drUnitID       : ARRAY [0..3] OF LONGCARD;
        drWaiting      : List;
        drDiscBlock    : Interrupt;
        drDiscSync     : Interrupt;
        drIndex        : Interrupt;

```

```

    END;

```

```

CONST

```

```

    (* Hardware Magic *)

```

```

    DskDMAOPF = 4000H; (* idle command for dsklen register *)

```

```

    DRA1locUnit = LibBase - 0*LibVectSize;

```

```

    DRFreeUnit = LibBase - 1*LibVectSize;

```

```

    DRGetUnit = LibBase - 2*LibVectSize;

```

```

    DRGiveUnit = LibBase - 3*LibVectSize;

```

```

    DRGetUnitID = LibBase - 4*LibVectSize;

```

```

    DRLastComm = DRGiveUnit; (* probably an error?? maybe DRGetUnitID *)

```

```

    (* drive types *)

```

```

    DRTAmiga = 000000000H;

```

```

    DRT3742202S = 055555555H;

```

```

    DRTEmpty = 0FFFFFFFH;

```

```

END DiskResource.

```

Module Display

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Display.DEF                      Version: Amiga.00.00
* Created: 11/30/86   Updated: 05/07/87   Author: Leon Frenkel
* Description: Amiga display control registers definitions.
*****)
```

DEFINITION MODULE Display;

CONST

```
(* bplcon0 register *)
Mode640      = 8000H;
PlnCntrlMsk  = 0007H; (* how many bit planes? 0 = non 1-> = 1->6, 7 = reserved*)
PlnCntrlShft = 12;    (* bits to shift for bplcon0 *)
PF2Pri       = 0040H; (* bplcon2 bit *)
ColorOn      = 0200H; (* disable color burst *)
Db1PF        = 0400H;
HoldnModifty = 0800H;
Interlace    = 0004H; (* interlace mode for 400 *)
```

```
(* bplcon1 register *)
PFAFineScroll = 000FH;
PFBFineScrollShift = 4;
PFFineScrollMask = 000FH;
```

```
(* display window start and stop *)
DIWHorizPos   = 007FH; (* horizontal start/stop *)
DIWVrtclPos   = 01FFH; (* vertical start/stop *)
DIWVrtclPosShift = 7;
```

```
(* Data fetch start/stop horizontal position *)
DFtchMask = 00FFH;
```

```
(* vposr bits *)
VPosrLOF = 8000H;
```

END Display.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****)
* Name: DMAHardware.DEF                      Version: Amiga.00.00
* Created: 11/29/86   Updated: 11/29/86   Author: Leon Frenkel
* Description: Amiga DMA control definitions.
*****)

DEFINITION MODULE DMAHardware;

(*$L+*)

CONST
  (* write definitions for dmaconw *)
  DMASetClr = 15;
  DMAAud0 = 0;
  DMAAud1 = 1;
  DMAAud2 = 2;
  DMAAud3 = 3;
  DMADisk = 4;
  DMASprite = 5;
  DMABlitter = 6;
  MACopper = 7;
  DMARaster = 8;
  DMAMaster = 9;
  DMABlitHog = 10;
  DMAAll = {0..8}; (* all dma channels *)
  DMAAudio = {0..3}; (* 4 bit mask *)

  (* read definitions for dmaconr *)
  (* bits 0-8 correspond to dmaconw definitions *)
  DMABltDone = 14;
  DMABltNZero = 13;

PROCEDURE OnDisplay;
(* Implementation of 'C' macro:
  * #define ON_DISPLAY      custom.dmacon = BITSET|DMAF_RASTER;
  *)

PROCEDURE OffDisplay;
(* Implementation of 'C' macro:
  * #define OFF_DISPLAY     custom.dmacon = BITCLR|DMAF_RASTER;
  *)

PROCEDURE OnSprite;
(* Implementation of 'C' macro:
  * #define ON_SPRITE      custom.dmacon = BITSET|DMAF_SPRITE;
  *)

PROCEDURE OffSprite;
(* Implementation of 'C' macro:
  * #define OFF_SPRITE     custom.dmacon = BITCLR|DMAF_SPRITE;
  *)

END DMAHardware.

```


Module Drawing

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Drawing.DEF                      Version: Amiga.00.00
* Created: 11/20/86   Updated: 05/07/87   Author: Leon Frenkel
* Description: Graphics Drawing functions.
*****)
```

DEFINITION MODULE Drawing;

```
FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Rasters IMPORT RastPort, DrawModeSet;
```

```
PROCEDURE BndryOff(VAR rp: RastPort);
(* Turn boundary mode off. This procedure is a 'C' macro equivalent:
* #define BNDRYOFF(w)    {(w)->Flags &= ~AREAOUTLINE;}
```

rp - RastPort *)

```
PROCEDURE ClearEOL(VAR rp: RastPort);
(* Clear from current position to end of line.
```

rp - RastPort structure *)

```
PROCEDURE ClearScreen(VAR rp: RastPort);
(* Clear from current position to end of RastPort.
```

rp - RastPort structure *)

```
PROCEDURE Draw(VAR rp: RastPort; x, y: INTEGER);
(* Draw a line between the current pen position and the new x,y position.
```

rp - RastPort structure
x - x component of end point of line
y - y component of end point of line *)

```
PROCEDURE DrawCircle(VAR rp: RastPort; cx, cy, radius: INTEGER);
(* Draw a circle centered at cx,cy with a specified radius. This function
* corresponds to the 'C' language macro. The 'C' macro is:
* #define DrawCircle(rp,cx,cy,r) DrawEllipse(rp,cx,cy,r,r);
```

rp - RastPort structure
cx - x coordinate of the centerpoint of the circle
cy - y coordinate of the centerpoint of the circle
radius - radius of the circle *)

```
PROCEDURE DrawEllipse(VAR rp: RastPort; cx, cy, a, b: INTEGER);
(* Draw an ellipse centered at cx,cy with vertical and horizontal radii
of a,b respectively.
```

rp - RastPort into which the ellipse will be drawn
cx - x coordinate of the centerpoint of the ellipse
cy - y coordinate of the centerpoint of the ellipse
a - the horizontal radius of the ellipse
b - the vertical radius of the ellipse *)

```
PROCEDURE Flood(VAR rp: RastPort; mode: LONGCARD; x, y: INTEGER);
(* Flood rastport like areafill.
```

rp - RastPort structure
mode - 0 = Border Fill, 1 = Fill all adjacement pixels that have the same
pen number as (x,y)
x - x component of flood fill starting point
y - y component of flood fill starting point *)


```
PROCEDURE Move(VAR rp: RastPort; x, y: INTEGER);
(* Move graphics pen position.
```

```
rp - RastPort structure
x - x component of new pen position
y - y component of new pen position *)
```

```
PROCEDURE PolyDraw(VAR rp: RastPort; count: INTEGER; array: ADDRESS);
(* Draw lines from table of (x,y) values.
```

```
rp - RastPort structure
count - number of points in array (x,y) pairs
array - pointer to first (x,y) pair *)
```

```
PROCEDURE ReadPixel(VAR rp: RastPort; x, y: INTEGER): INTEGER;
(* Read the pen number value of the pixel at a specified x,y location
within a certain RastPort.
```

```
rp - RastPort structure
x - x component of point
y - y component of point

result - (0..255) number at the position is returned. -1 is returned
if cannot read that pixel *)
```

```
PROCEDURE RectFill(VAR rp: RastPort; xmin, ymin, xmax, ymax: INTEGER);
(* Fill a defined rectangular area with the current drawing pen color,
outline color, secondary color, and pattern.
```

```
rp - RastPort structure
xmin - x component of upper left corner
ymin - y component of upper left corner
xmax - x component of lower right corner
ymax - y component of lower right corner *)
```

```
PROCEDURE SetAPen(VAR rp: RastPort; pen: CARDINAL);
(* Set primary pen.
```

```
rp - RastPort structure
pen - (0..255) *)
```

```
PROCEDURE SetAPt(VAR rp: RastPort; p: ADDRESS; n: CARDINAL);
(* Set area pattern. This procedure corresponds to the 'C' macro:
* #define SetAPt(w,p,n) {(w)->AreaPtrn = p;(w)->AreaPtSz = n;}
```

```
rp - RastPort structure
p - pointer to area pattern
n - size of area pattern *)
```

```
PROCEDURE SetBPen(VAR rp: RastPort; pen: CARDINAL);
(* Set secondary pen.
```

```
rp - pointer to RastPort structure
pen - (0..255) *)
```

```
PROCEDURE SetDrMd(VAR rp: RastPort; mode: DrawModeSet);
(* Set drawing mode.
```

```
rp - pointer to RastPort structure
mode - Jam1, Jam2, ... some combinations may not make much sense *)
```

Module Drawing

PROCEDURE SetDrPt(VAR rp: RastPort; p: BITSET);

(* Set line pattern. This procedure corresponds to a 'C' macro:

* #define SetDrPt(w,p)

* {(w)->LinePtrn = p;(w)->Flags |= FRST_DOT;(w)->linpatcnt=15;}

rp - RastPort structure

p - line pattern *)

PROCEDURE SetOPen(VAR rp: RastPort; pen: CARDINAL);

(* Change the area outline pen and turn on outline mode for areafills.

* This procedure corresponds to the 'C' macro:

* #define SetOPen(w,c) {(w)->AO1Pen = c;(w)->Flags |= AREAOUTLINE;}

rp - pointer to RastPort structure

pen - number between (0..255) *)

PROCEDURE SetWrMsk(VAR rp: RastPort; m: BYTE);

(* Set rast port write mask. Corresponds to 'C' macro:

* #define SetWrMsk(w,m) {(w)->Mask = m;}

rp - RastPort structure

m - mask (any 8-bit type) *)

PROCEDURE WritePixel(VAR rp: RastPort; x, y: INTEGER): INTEGER;

(* Change the pen num of one specific pixel in a specified RastPort.

rp - RastPort structure

x - x component of point to set

y - y component of point to set

result - 0 if pixel successfully changed, -1 if (x,y) is outside the RastPort*)

END Drawing.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Exec.DEF                      Version: Amiga.00.00      *
* Created: 11/19/86   Updated: 07/24/87   Author: Leon Frenkel  *
* Description: Amiga ROM Kernel Executive.                  *
*****)

```

```
DEFINITION MODULE Exec;
```

```
FROM SYSTEM IMPORT ADDRESS;
```

```
CONST
```

```
  ExecName = "exec.library";
```

```
PROCEDURE Debug;
```

```
(* Run the system debugger. *)
```

```
PROCEDURE GetCC(): BITSET;
```

```
(* Get condition codes in a 68010 compatible way.
```

```
  result - the 68000/68010 condition codes *)
```

```
PROCEDURE RawDoFmt(FormatString: ADDRESS; DataStream: ADDRESS;
```

```
  PutChProc: ADDRESS; PutChData: ADDRESS);
```

```
(* Format data into a character stream. Perform "C"-language-like formatting  
of a data stream, output the result a character at a time.
```

```
  FormatString - a "C"-language-like null terminated format string,  
with '%' formatting info.
```

```
  DataStream - a stream of data that is interpreted according to the format str
```

```
  PutChProc - the procedure to call with each character to be output, called:
```

```
    PutChProc( Char, PutChData);
```

```
    DO-0:8 A3
```

```
the procedure is called with a null Char at the end of the format  
string.
```

```
  PutChData - an address register that passes thru to PutChProc *)
```

```
PROCEDURE SetSR(newSR, mask: BITSET): BITSET;
```

```
(* Get and/or set processor status register.
```

```
  newSR - new values for bits specified in the mask, other bits are not effected
```

```
  mask - bits to be changed
```

```
  result - the entire status register before new bits *)
```

```
PROCEDURE SumKickData;
```

```
(* Compute the checksum for the kickstart delta list. *)
```

```
END Exec.
```

Module ExecBase

```
(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: ExecBase.DEF           Version: Amiga.00.00
* Created: 11/22/86   Updated: 07/24/87   Author: Leon Frenkel
* Description: Exec Library Base.
*****)
```

DEFINITION MODULE ExecBase;

```
FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM Interrupts IMPORT IntVector, SoftIntList;
FROM Libraries IMPORT Library;
FROM Lists IMPORT List;
FROM Tasks IMPORT SignalSet, TrapSet, TaskPtr;
```

CONST

```
(* ExecBase.AttnFlags *)
(* Processor and Co-processors *)
AF68010 = 0; (* also set for 68020 *)
AF68020 = 1;
AF68881 = 4;
```

TYPE

```
ExecBasePtr = POINTER TO ExecBase;
```

```
ExecBase =
```

```
RECORD
```

```
  LibNode          : Library;
```

```
  SoftVer          : CARDINAL; (* kickstart release numbre *)
  LowMemChkSum     : INTEGER;
  ChkBase          : LONGCARD; (* system base pointer complement *)
  ColdCapture      : ADDRESS;  (* coldstart soft vector *)
  CoolCapture      : ADDRESS;
  WarmCapture      : ADDRESS;
  SysStkUpper      : ADDRESS;  (* system stack base (upper bound) *)
  SysStkLower      : ADDRESS;  (* top of system stack (lower bound) *)
  MaxLocMem        : LONGCARD;
  DebugEntry       : ADDRESS;
  DebugData        : ADDRESS;
  AlertData        : ADDRESS;
  MaxExtMem        : ADDRESS;  (* top of extended mem or null if none *)
```

```
  ChkSum           : CARDINAL;
```

```
(* Interrupt related *)
```

```
IntVects          : ARRAY [0..15] OF IntVector;
```

```
(* System Variables *)
```

```
ThisTask          : TaskPtr;  (* pointer to current task *)
IdleCount          : LONGCARD; (* idle counter *)
DispCount          : LONGCARD; (* dispatch counter *)
Quantum            : CARDINAL; (* time slice quantum *)
Elapsed            : CARDINAL; (* current quantum ticks *)
SysFlags           : BITSET;  (* misc system flags *)
IDNestCnt          : BYTE;    (* interrupt disable nesting count *)
TDNestCnt          : BYTE;    (* task disable nesting count *)
```

```
AttnFlags          : BITSET;  (* special attention flags *)
AttnResched        : BITSET;  (* rescheduling attention *)
ResModules         : ADDRESS;  (* resident module array pointer *)
```

```
TaskTrapCode       : ADDRESS;
TaskExceptCode      : ADDRESS;
TaskExitCode        : ADDRESS;
TaskSigAlloc        : SignalSet;
TaskTrapAlloc       : TrapSet;
```

```

(* System Lists *)
MemList      : List;
ResourceList : List;
DeviceList   : List;
IntrList     : List;
LibList      : List;
PortList     : List;
TaskReady    : List;
TaskWait     : List;

SoftInts      : ARRAY [0..4] OF SoftIntList;

(* Other Globals *)
LastAlert     : ARRAY [0..3] OF LONGINT;

(* these next two variables are provided to allow
 * system developers to have a rough idea of the
 * period of two externally controlled signals --
 * the time between vertical blank interrupts and the
 * external line rate (which is counted by CIA A's
 * "time of day" clock). In general these values
 * will be 50 or 60, and may or may not track each
 * other. These values replace the obsolete AFB_PAL
 * and AFB_50HZ flags.
 *)
VBlankFrequency : BYTE;
PowerSupplyFrequency : BYTE;

SemaphoreList : List;

(* these next two are to be able to kickstart into user ram.
 * KickMemPtr holds a singly linked list of MemLists which
 * will be removed from the memory list via AllocAbs. If
 * all the AllocAbs's succeeded, then the KickTagPtr will
 * be added to the rom tag list.
 *)
KickMemPtr      : ADDRESS; (* ptr to queue of mem lists *)
KickTagPtr      : ADDRESS; (* ptr to rom tag queue *)
KickChecksum    : ADDRESS; (* checksum for mem and tags *)

ExecBaseReserved : ARRAY [0..9] OF BYTE;
ExecBaseNewReserved : ARRAY [0..19] OF BYTE;
END;

END ExecBase.

```

Module Expansion

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1987 by Leon Frenkel                *
*                               *
*****
* Name: Expansion.DEF                      Version: Amiga.00.00      *
* Created: 07/27/87   Updated: 07/27/87   Author: Leon Frenkel      *
* Description: Definitions for expansion.library                      *
*****)
```

DEFINITION MODULE Expansion;

```
FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM ConfigRegs IMPORT ExpansionRom;
FROM FileHandler IMPORT DeviceNodePtr;
FROM Libraries IMPORT LibraryPtr;
FROM Nodes IMPORT Node;
```

```
CONST
  ExpansionName = "expansion.library";
```

```
VAR
  ExpansionBase : LibraryPtr;
```

```
CONST
  (* flags for the AddDosNode() call *)
  ADNStartProc = 0;
```

```
(* cdFlags *)
CDShutup   = 0; (* this board has been shut up *)
CDConfigMe = 1; (* this board needs a driver to claim it *)
```

TYPE

```
ConfigDevPtr = POINTER TO ConfigDev;
ConfigDev = RECORD
  cdNode      : Node;
  cdFlags     : BYTE;
  cdPad       : BYTE;
  cdRom       : ExpansionRom; (* image of expansion rom area *)
  cdBoardAddr : ADDRESS;      (* where in memory the board is *)
  cdBoardSize : ADDRESS;      (* size in bytes *)
  cdSlotAddr  : CARDINAL;     (* which slot number *)
  cdSlotSize  : CARDINAL;     (* number of slots the board takes *)
  cdDriver    : ADDRESS;      (* pointer to node of driver *)
  cdNextCD    : ConfigDevPtr; (* linked list of drivers to config *)
  cdUnused    : ARRAY [0..3] OF LONGCARD; (* for whatever driver wants *)
END;
```

(* this structure is used by GetCurrentBinding() and SetCurrentBinding() *)

```
CurrentBindingPtr = POINTER TO CurrentBinding;
CurrentBinding = RECORD
  cbConfigDev : ConfigDevPtr; (* first configdev in chain *)
  cbFileName  : ADDRESS;      (* file name of driver *)
  cbProductString : ADDRESS;  (* product # string *)
  cbToolTypes : POINTER TO ADDRESS; (* tooltypes from disk object *)
END;
```

```
PROCEDURE AddDosNode(bootPri: INTEGER; flags: BITSET;
  deviceNode: DeviceNodePtr): CARDINAL;
(* Mount a disk to the system.
```

bootPri - a BYTE quantity with the boot priority for this disk
flags - flag bits: {ADNStartProc} starts a handler process immediately
deviceNode - a legal DOS device node, properly initialized or NIL

result - non-zero everything went ok, zero if we ran out of memory *)

```
PROCEDURE MakeDosNode(parameterPkt: ADDRESS): DeviceNodePtr;
(* Construct dos data structure that a disk needs.
```

parameterPkt - a longword array containing all the information needed to
initialize data structures.

longword	description
0	string with dos handler name
1	string with exec device name
2	unit number (for OpenDevice)
3	flags (for OpenDevice)
4	# of longwords in rest of enviroment
5-n	file handler environment (see libraries/filehandler.h)

result - pointer to initied device node or NIL if not enough memory *)

PROCEDURE AddConfigDev(VAR configDev: ConfigDev);
(* Add a new ConfigDev structure to the system.

configDev - a valid ConfigDev structure *)

PROCEDURE AllocBoardMem(slotSpec: INTEGER): INTEGER;
(* Allocate standard expansion memory.

slotSpec - the memory size field of the Type byte of an expansion board

result - the slot number that was allocated, or -1 for error *)

PROCEDURE AllocConfigDev(): ConfigDevPtr;
(* Allocate a ConfigDev structure

result - either a valid ConfigDev structure or NIL *)

PROCEDURE AllocExpansionMem(numSlots, slotOffset: INTEGER): INTEGER;
(* Allocate expansion memory.

numSlots - the number of slots required

slotOffset - an offset from the boundary for startSlot

startSlot - the slot number that was allocate, or -1 for error *)

PROCEDURE ConfigBoard(board: ADDRESS; VAR configDev: ConfigDev): CARDINAL;
(* Configure a board.

board - the current address that the expansion board is responding

configDev - an initialized ConfigDev structure

result - non-zero if there was a problem configuring this board *)

PROCEDURE ConfigChain(baseAddr: ADDRESS): CARDINAL;
(* Configure the whole system.

baseAddr - the base address to start looking for boards

result - non-zero if something went wrong *)

PROCEDURE FindConfigDev(oldConfigDev: ConfigDevPtr; manufacturer: LONGINT;
product: LONGINT): ConfigDevPtr;
(* Find a matching ConfigDev entry.

oldConfigDev - a valid ConfigDev struc, or NIL to start from start of list

manufacturer - the manufacturer code being searched for, -1 to ignore num

product - the product code being searched for, -1 to ignore product num

result - next ConfigDev entry that matches or NIL if no more matches *)

PROCEDURE FreeBoardMem(startSlot, slotSpec: INTEGER);
(* Allocate standard device expansion memory.

startSlot - a slot number in expansion space

slotSpec - the memory size field of the Type byte of an expansion board *)

Module Expansion

```
PROCEDURE FreeConfigDev(VAR configDev: ConfigDev);  
(* Free a ConfigDev structure.
```

configDev - a valid ConfigDev structure *)

```
PROCEDURE FreeExpansionMem(startSlot: INTEGER);  
(* Free standard device expansion memory.
```

startSlot - the slot number that was allocated
numSlots - the number of slots to be freed *)

```
PROCEDURE GetCurrentBinding(VAR currentBinding: CurrentBinding;  
size: CARDINAL): CARDINAL;  
(* Sets static board configuration area.
```

currentBinding - variable structure to be filled in
size - size of the user's binddriver structure

result - the true size of a CurrentBinding structure is returned *)

```
PROCEDURE ObtainConfigBinding();  
(* Try to get permission to bind drivers. *)
```

```
PROCEDURE ReadExpansionByte(board: ADDRESS; offset: LONGCARD): INTEGER;  
(* Read a byte nybble by nybble.
```

board - a pointer to the base of a new style expansion board
offset - a logical offset from the board base

result - a byte of data from the expansion board or -1 if error *)

```
PROCEDURE ReadExpansionRom(board: ADDRESS; VAR configDev: ConfigDev): INTEGER;  
(* Read a boards configuration rom space.
```

board - a pointer to the base of a new style expansion board
configDev - the ConfigDev structure to be filled in

result - non-zero if board address does not contain valid expansion board *)

```
PROCEDURE ReleaseConfigBinding();  
(* Allow others to bind to drivers. *)
```

```
PROCEDURE RemConfigDev(VAR configDev: ConfigDev);  
(* Remove a ConfigDev structure from the system.
```

configDev - a valid ConfigDev structure *)

```
PROCEDURE SetCurrentBinding(VAR currentBinding: CurrentBinding; size: CARDINAL);  
(* Sets static board configuration area.
```

currentBinding - a CurrentBinding structure
size - size of the user's binddriver structure *)

```
PROCEDURE WriteExpansionByte(board: ADDRESS; offset: LONGCARD;  
byte: BYTE): INTEGER;  
(* Write a byte nybble by nybble.
```

board - pointer to the base of a new style expansion board
offset - a logical offset from the configdev base
byte - a byte of data to be written to the expansion board

result - zero if success, non-zero there was a problem *)

END Expansion.


```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: FileHandler.DEF          Version: Amiga.00.00
* Created: 07/27/87   Updated: 07/27/87   Author: Leon Frenkel
* Description: Device and file handler specific code for AmigaDOS.
*****)

DEFINITION MODULE FileHandler;

FROM SYSTEM IMPORT ADDRESS;
FROM AmigaDOS IMPORT BPTR, BSTR;

(* The disk "environment" is a longword array that describes the
 * disk geometry. It is variable sized, with the length at the beginning.
 * Here are the constants for a standard geometry.
 *)

CONST
  (* these are the offsets into the array *)
  DETableSize    = 0; (* standard value is 11 *)
  DESizeBlock    = 1; (* in longwords: standard value is 128 *)
  DESecOrg       = 2; (* not used; must be 0 *)
  DENumHeads     = 3; (* # of heads (surfaces). drive specific *)
  DESecsPerBlk   = 4; (* not used; must be 1 *)
  DEBlksPerTrack = 5; (* blocks per track. drive specific *)
  DEReservedBlks = 6; (* unavailable blocks at start. usually 2 *)
  DEPrefac       = 7; (* not used; must be 0 *)
  DEInterleave   = 8; (* usually 0 *)
  DELowCyl       = 9; (* starting cylinder, typically 0 *)
  DEUpperCyl     = 10; (* max cylinder. drive specific *)
  DENumBuffers   = 11; (* starting # of buffers. typically 5 *)
  DEMemBufType   = 12; (* type of mem to allocate for buffers.
                        * default is 3, hard disk want 0 *)

TYPE

(* The file system startup message is linked into a device node's startup
 * field. It contains a pointer to the above environment, plus the
 * information needed to do an exec OpenDevice().
 *)
FileSysStartupMsgPtr = POINTER TO FileSysStartupMsg;
FileSysStartupMsg = RECORD
  fssmUnit   : LONGCARD; (* exec unit number for device *)
  fssmDevice : BSTR;      (* null term bstring to device name *)
  fssmEnviron : BPTR;     (* ptr to environment table *)
  fssmFlags  : LONGCARD; (* flags for OpenDevice() *)
END;

(* The module "AmigaDOSExt" has a DeviceList structure.
 * The "device list" can have one of three different things linked onto
 * it. Dosextens defines the structure for a volume. DLT_DIRECTORY
 * is for an assigned directory. The following structure is for
 * a dos "device" (DLT_DEVICE).
 *)

```

Module FileHandler

DeviceNodePtr = POINTER TO DeviceNode;

DeviceNode = RECORD

```

    dnNext      : BPTR;      (* singly linked list *)
    dnType      : LONGCARD;  (* always 0 for dos "devices" *)
    dnTask      : ADDRESS;   (* MsgPortPtr.
                             * standard dos task field. If this is
                             * null when the node is accessed,
                             * a task will be started up
                             *)
    dnLock      : BPTR;      (* not used for devices -- leave null *)
    dnHandler    : BSTR;     (* filename to loadseg (if seglist null)*)
    dnStackSize : LONGCARD;  (* stacksize to use when starting task *)
    dnPriority   : LONGINT;   (* task priority when starting task *)
    dnStartup   : BPTR;      (* startup msg: FileSysStartupMsg for disks *)
    dnSegList   : BPTR;      (* code to run to start new task
                             * if null then dnHandler will be loaded
                             *)
    dnGlobalVec : BPTR;      (* BCPL global vector to use when starting
                             * a task. -1 means that dnSegList is not
                             * for a bcpl program, so the dos won't
                             * try and construct one. 0 tell the
                             * dos that you obey BCPL linkage rules,
                             * and that it should construct a global
                             * vector for you.
                             *)
    dnName      : BSTR;      (* the node name, e.g. '\3','D','F','3' *)
END;
```

END FileHandler.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*****
* Name: FileSystem.DEF              Version: Amiga.00.00      *
* Created: 11/19/86   Updated: 02/08/87   Author: Leon Frenkel *
* Description: This module represents file I/O at a level above the *
*   AmigaDOS module. This module is described in appendix 2 of Programming *
*   in Modula-2 by N. Wirth. For purposes of efficiency this module performs *
*   it own buffering. This resultst in much faster I/O operations.      *
*****)

DEFINITION MODULE FileSystem;

FROM SYSTEM IMPORT ADDRESS, WORD;
FROM AmigaDOS IMPORT FileHandle;

(*$L+*)

VAR
  (* The size of the buffer to allocate for next file to be opened *)
  BufferSize : LONGCARD;

TYPE
  Response = (done,          (* previous operation completed successfully *)
             notdone,        (* previous operation failed *)
             nomemory);      (* Lookup() failed because not enough mem for buffer *)

  FilePtr = POINTER TO File;
  File = RECORD
    res          : Response;  (* set to done if no error *)
    eof          : BOOLEAN;   (* TRUE if end of file encountered *)
    err          : LONGINT;   (* if notdone then AmigaDOS error code *)
    handle       : FileHandle; (* AmigaDOS file handle for this file *)
    filePos      : LONGCARD;  (* current pos of AmigaDOS file ptr *)
    bufferPtr    : ADDRESS;   (* ptr to first byte of buffer *)
    bufferSize   : LONGCARD;  (* Size of buffer in bytes *)
    bufferBase   : LONGCARD;  (* actual file pos of first byte of buf*)
    bufferOffset : LONGCARD;  (* current offset into buffer *)
    bufferEnd    : LONGCARD;  (* end of buffer offset *)
    bufferChanged : BOOLEAN;  (* buffer has been written to *)
  END;

PROCEDURE Close(VAR f: File);
(* Close an open file.

   f - File structure *)

PROCEDURE Delete(filename: ARRAY OF CHAR);
(* Delete a file of a specified name.

   filename - name of file to be deleted *)

PROCEDURE Lookup(VAR f: File; filename: ARRAY OF CHAR; new: BOOLEAN);
(* Open an old file or a new file.

   f - File structure
   filename - name of file to open
   new - if FALSE then open an existing file. if TRUE open a new file *)

PROCEDURE SetPos(VAR f: File; pos: LONGCARD);
(* Move file ptr to a new position.

   f - File structure
   pos - new file ptr position *)

```

Module FileSystem

```
PROCEDURE GetPos(VAR f: File; VAR pos: LONGCARD);
(* Return the current file ptr position.
```

```
  f - File structure
  pos - variable into which the current file ptr position is to be stored *)
```

```
PROCEDURE Length(VAR f: File; VAR length: LONGCARD);
(* Return the length of a file.
```

```
  f - File structure
  length - variable into which the length of the file is to be stored *)
```

```
PROCEDURE ReadWord(VAR f: File; VAR w: WORD);
(* Read a word(16-bits) of data from a file.
```

```
  f - File structure
  w - variable into which the data is to be stored *)
```

```
PROCEDURE WriteWord(VAR f: File; w: WORD);
(* Write a word(16-bits) of data to a file.
```

```
  f - File structure
  w - a word value to be written *)
```

```
PROCEDURE ReadChar(VAR f: File; VAR ch: CHAR);
(* Read a char(8-bits) of data from a file.
```

```
  f - File structure
  ch - variable into which the data is to be stored *)
```

```
PROCEDURE WriteChar(VAR f: File; ch: CHAR);
(* Write a char(8-bits) of data to a file.
```

```
  f - File structure
  ch - a char value to be written *)
```

```
END FileSystem.
```

```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: GamePortDevice.DEF                               Version: Amiga.00.00
* Created: 11/26/86   Updated: 12/30/86   Author: Leon Frenkel
* Description: "gameport.device" types.
*****)

```

```

DEFINITION MODULE GamePortDevice;

```

```

FROM IODevices IMPORT CmdNonStd;

```

```

CONST

```

```

(* GamePort commands *)
GPDReadEvent = CmdNonStd + 0;
GPDAaskCType = CmdNonStd + 1;
GPDSsetCType = CmdNonStd + 2;
GPDAaskTrigger = CmdNonStd + 3;
GPDSsetTrigger = CmdNonStd + 4;

```

```

TYPE

```

```

(* GamePort structures *)

```

```

(* gptKeys *)

```

```

GPTKeys = (GPTDownKeys,
            GPTUpKeys,
            GPT2, GPT3, GPT4, GPT5, GPT6, GPT7, GPT8, GPT9,
            GPT10, GPT11, GPT12, GPT13, GPT14, GPT15);
GPTKeysSet = SET OF GPTKeys;

```

```

GamePortTriggerPtr = POINTER TO GamePortTrigger;

```

```

GamePortTrigger = RECORD

```

```

    gptKeys : GPTKeysSet; (* key transition triggers *)
    gptTimeout : CARDINAL; (* time trigger (vblank units) *)
    gptXDelta : CARDINAL; (* X distance trigger *)
    gptYDelta : CARDINAL; (* Y distance trigger *)
END;

```

```

CONST

```

```

(* Controller Types *)

```

```

GPCTAllocated = -1; (* allocate by another user *)
GPCTNoController = 0;

```

```

GPCTMouse = 1;
GPCTRelJoystick = 2;
GPCTAbsJoystick = 3;

```

```

(* Errors *)

```

```

GPDErrSetCType = 1; (* this controller not valid at this time *)

```

```

END GamePortDevice.

```

Module Gels

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Gels.DEF                      Version: Amiga.00.00      *
* Created: 11/20/86   Updated: 05/09/87   Author: Leon Frenkel  *
* Description: Graphics Elements Animation types/functions.      *
*****)

```

DEFINITION MODULE Gels;

```

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Rasters IMPORT RastPort;
FROM Views IMPORT ViewPort;

```

TYPE

```

VSpriteFlags = (* User-set VSprite flags *)
  (VSprite, (* set if VSprite, clear if Bob *)
   SaveBack, (* set if background is to be saved/restored *)
   Overlay, (* set to mask image of Bob onto background *)
   MustDraw, (* set if VSprite absolutely must be drawn *)
   VSF4, (* Unmapped user bits 4..7 *)
   VSF5,
   VSF6,
   VSF7,
   (* System-set VSprite flags *)
   BackSaved, (* this Bob's background has been saved *)
   BobUpdate, (* temporary flag, useless to outside world *)
   GelGone, (* set if gel is completely clipped (offscreen) *)
   VSOverflow, (* VSprite overflow (if MustDraw set we draw!) *)
   VSF12, (* Unmapped system bits 12..15 *)
   VSF13,
   VSF14,
   VSF15);

```

VSpriteFlagsSet = SET OF VSpriteFlags;

CONST

```

(* Mask of all user settable VSprite-flags *)
SUserFlags = VSpriteFlagsSet{VSprite..VSF7};

```

TYPE

```

BobFlags = (* these are the user flag bits *)
  (SaveBob, (* set to not erase Bob *)
   BobIsComp, (* set to identify Bob as AnimComp *)
   BF2, (* Unmapped bob user flags 2..7 *)
   BF3,
   BF4,
   BF5,
   BF6,
   BF7,
   (* these are the system flag bits *)
   BWaiting, (* set while Bob is waiting on 'after' *)
   BDrawn, (* set when Bob is drawn this DrawG pass *)
   BobsAway, (* set to initiate removal of Bob *)
   BobNix, (* set when Bob is completely removed *)
   SavePreserve, (* for back-restore during double-buffer *)
   OutStep, (* for double-clearing if double-buffer *)
   BF14, (* Unmapped system bob flags 14..15 *)
   BF15);

```

BobFlagsSet = SET OF BobFlags;

CONST

```

(* Mask of all user settable Bob-flags *)
BUserFlags = BobFlagsSet{SaveBob..BF7};

```

(* defines for the animation procedures *)

```

AnFracSize = 6;
AnimHalf = 0020H;

```

```

(* AnimComp.Flags = RingTrigger *)
RingTrigger = {0}; (* BITSET definition *)

```

TYPE

(* UserStuff definitions *)

```

(* Store pointers pointer to user extension structure. *)
VUserStuff = ADDRESS; (* VSprite user stuff *)
BUserStuff = ADDRESS; (* Bob user stuff *)
AUserStuff = ADDRESS; (* AnimOb user stuff *)

(* Forward pointers definitions *)
VSpritePtr = POINTER TO VSprite;
BobPtr = POINTER TO Bob;
AnimObPtr = POINTER TO AnimOb;
AnimCompPtr = POINTER TO AnimComp;
DBufPacketPtr = POINTER TO DBufPacket;
GelsInfoPtr = POINTER TO GelsInfo;

(**** Gel Structures ****)
VSprite = RECORD
    (* GEL linked list forward/backward ptrs sorted by y,x value *)
    NextVSprite : VSpritePtr;
    PrevVSprite : VSpritePtr;

    (* GEL draw list constructed in the order the Bobs are actually
    * drawn, then list is copied to clear list must be here in
    * VSprite for system boundary detection.
    *)
    DrawPath : VSpritePtr; (* pointer of overlay drawing *)
    ClearPath : VSpritePtr; (* pointer for overlay drawing *)

    (* The VSprite positions are defined in (y,x) order to make
    * sorting easier, since (y,x) as a long integer
    *)
    OldY : INTEGER; (* previous position *)
    OldX : INTEGER;

    (* COMMON VARIABLES *)
    Flags : VSpriteFlagsSet; (* VSprite flags *)

    (* USER VARIABLES *)
    Y : INTEGER; (* screen position *)
    X : INTEGER;

    Height : INTEGER;
    Width : INTEGER; (* number of words per row of image data *)
    Depth : INTEGER; (* number of planes of data *)

    MeMask : BITSET; (* which types can collide with this VSprite *)
    HitMask : BITSET; (* which types this VSprite can collide with *)

    ImageData : ADDRESS; (* pointer to VSprite image *)

    (* BorderLine is the one-dimensional logical OR of all the
    * VSprite bits, used for fast collision detection of edge
    *)
    BorderLine : ADDRESS; (* logical OR of all VSprite bits *)
    CollMask : ADDRESS; (* similar to above except this is a matrix *)

    (* Pointer to this VSprite's color definitions (not used by Bobs) *)
    SprColors : ADDRESS;

    VSBob : BobPtr; (* points home if this VSprite is part of a Bob *)

    (* planePick flag: set bit selects a plane from image, clear bit selects
    * use of shadow mask for that plane
    * OnOff flag: if using shadow mask to fill plane, this bit (corresponding
    * to bit in planePick) describes whether to fill with 0's or 1's
    * There are two uses for these flags:
    * - if this is the VSprite of a Bob, these flags describe how the Bob
    * is to be drawn into memory
    * - if this is a simple VSprite and the user intends on setting the
    * MustDraw flag of the VSprite, these flags must be set too to describe
    * which color registers the user wants for the image
    *)
    PlanePick : BYTE;
    PlaneOnOff : BYTE;

    VUserExt : VUserStuff; (* User definable: see note above *)
END;

```


Module Gels

```
(* Blitter-Objects *)
Bob = RECORD
    (* COMMON VARIABLES *)
    Flags      : BobFlagsSet; (* general purpose flags (see above) *)

    (* USER VARIABLES *)
    SaveBuffer : ADDRESS; (* pointer to the buffer for background save *)

    (* used by Bob for "cookie-cutting" and multi-plane masking *)
    ImageShadow : ADDRESS;

    (* pointer to BOBs for sequenced drawing of Bobs for current
    * overlying of multiple component animations
    *)
    Before      : BobPtr; (* draw this Bob before Bob pointed to by before *)
    After       : BobPtr; (* draw this Bob after Bob pointed to by after*)

    BobVSprite  : VSpritePtr; (* this Bob's VSprite definition *)

    BobComp     : AnimCompPtr; (* pointer to this Bob's AnimComp def *)

    DBuffer     : DBufPacketPtr; (* pointer to this Bob's dBuf packet *)

    BUserExt    : BUserStuff; (* Bob user extension *)
END;

AnimComp = RECORD
    (* COMMON VARIABLES *)
    Flags      : BITSET; (* AnimComp flags for system & user *)

    (* timer defines how long to keep this component active:
    * if set non-zero, timer decrements to zero then switches to nextSeq
    * if set to zero, AnimComp never switches
    *)
    Timer      : INTEGER;

    (* USER VARIABLES *)
    (* initial value for timer when the AnimComp is activated by the system *)
    TimeSet    : INTEGER;

    (* pointer to next and previous components of animation object *)
    NextComp   : AnimCompPtr;
    PrevComp   : AnimCompPtr;

    (* pointer to component definition of next image in sequence *)
    NextSeq    : AnimCompPtr;
    PrevSeq    : AnimCompPtr;

    AnimCRoutine : ADDRESS; (* special animation procedure *)

    YTrans     : INTEGER; (* initial y translation, if component*)
    XTrans     : INTEGER; (* initial x translation, if component*)

    HeadOb     : AnimObPtr;

    AnimBob    : BobPtr;
END;

AnimOb = RECORD
    (* SYSTEM VARIABLES *)
    NextOb     : AnimObPtr;
    PrevOb     : AnimObPtr;

    (* number of calls to Animate this AnimOb has endured *)
    Clock      : LONGINT;

    AnOldY     : INTEGER; (* old y,x coordinates *)
    AnOldX     : INTEGER;

    (* COMMON VARIABLES *)
    Any        : INTEGER; (* y,x coordinates of the AnimOb *)
```



```

AnX          : INTEGER;

(* USER VARIABLES *)
YVel         : INTEGER;      (* velocities of this object *)
XVel         : INTEGER;
YAccel       : INTEGER;      (* accelerations of this object *)
XAccel       : INTEGER;

RingYTrans   : INTEGER;      (* ring translation values *)
RingXTrans   : INTEGER;

AnimORoutine : ADDRESS;      (* special animation procedure *)

HeadComp     : AnimCompPtr;  (* pointer to first component *)

AUserExt     : AUserStuff;   (* AnimOb user extension *)
END;

(* dBufPacket defines the values needed to be saved across buffer to buffer
 * when in double-buffer mode
 *)
DBufPacket = RECORD
    BufY      : INTEGER;      (* save the other buffers screen coordinates *)
    BufX      : INTEGER;
    BufPath   : VSpritePtr;   (* carry the draw path over the gap *)

    (* these pointers must be filled in by the user *)
    (* pointer to other buffer's background save buffer *)
    BufBuffer : ADDRESS;
END;

(* a structure to contain the 16 collision procedure addresses *)
collTablePtr = POINTER TO collTable;
collTable = RECORD
    collPtrs : ARRAY [0..15] OF ADDRESS;
END;

GelsInfo = RECORD
    (* flag of which sprites to reserve from vsprite system *)
    sprRsrvd  : BYTE;
    Flags     : BYTE;        (* system use *)
    gelHead   : VSpritePtr;  (* dummy vSprites for list management *)
    gelTail   : VSpritePtr;
    (* pointer to array of 8 WORDS for sprite available lines *)
    nextLine  : ADDRESS;
    (* pointer to array of 8 pointers for color-last-assigned to vSprites *)
    lastColor : ADDRESS;
    collHandler : collTablePtr; (*addresses of collision routines*)
    leftmost  : INTEGER;
    rightmost : INTEGER;
    topmost   : INTEGER;
    bottommost : INTEGER;
    firstBlissObj : ADDRESS; (* system use only *)
    lastBlissObj : ADDRESS;
END;

CONST
    B2Norm    = 0;
    B2Swap    = 1;
    B2Bobber  = 2;

(* These bit descriptors are used by the GEL collide routines.
 * These bits are set in the hitMask and meMask variables of
 * a GEL to describe whether or not these types of collisions
 * can affect the GEL. BNDRY_HIT is described further below;
 * this bit is permanently assigned as the boundary-hit flag.
 * The other bit GEL_HIT is meant only as a default to cover
 * any GEL hitting any other; the user may redefine this bit.
 *)
BorderHit = 0;

TYPE
    (* Boundary Collision Mask *)

```

Module Gels

BoundaryColMask = SET OF [0..31];

CONST

```
(* These bit descriptors are used by the GEL boundry hit routines.
 * When the user's boundry-hit routine is called (via the argument
 * set by a call to SetCollision) the first argument passed to
 * the user's routine is the address of the GEL involved in the
 * boundry-hit, and the second argument has the appropriate bit(s)
 * set to describe which boundry was surpassed
 * VAR mask : BoundaryColMask;
 * IF TopHit IN mask THEN ...
 * ELSIF BottomHit IN mask THEN ...
 *)
TopHit    = 0;
BottomHit = 1;
LeftHit   = 2;
RightHit  = 3;
```

TYPE

```
(* User Procedure Called by Gel Software *)
BoundaryColProc = PROCEDURE(BoundaryColMask, VAR VSprite);
GelColProc      = PROCEDURE(VAR VSprite, VAR VSprite);
```

```
(* User Procedures Called by Animation Software *)
AnimCompProc   = PROCEDURE(VAR AnimComp);
AnimObProc     = PROCEDURE(VAR AnimOb);
```

```
PROCEDURE AddAnimOb(VAR anOb: AnimOb; VAR anKey: AnimObPtr; VAR rp: RastPort);
(* Add an AnimOb to the linked list of AnimObs.
```

```
anOb - an AnimOb structure to be added to the list
anKey - address of a pointer to the first AnimOb in the list
       (anKey = NIL if there are no AnimObs in the list so far)
rp - a valid RastPort *)
```

```
PROCEDURE AddBob(VAR bob: Bob; VAR rp: RastPort);
(* Adds a Bob to current gel list.
```

```
Bob - a Bob structure to be added to the gel list
rp - a RastPort structure *)
```

```
PROCEDURE AddVSprite(VAR vs: VSprite; VAR rp: RastPort);
(* Add a VSprite to the current gel list.
```

```
vs - the VSprite structure to be added to the gel list
rp - a RastPort structure *)
```

```
PROCEDURE Animate(VAR anKey: AnimObPtr; VAR rp: RastPort);
(* Processes every AnimOb in the current animation list.
```

```
anKey - address of the variable that points to the head AnimOb
rp - the RastPort structure *)
```

```
PROCEDURE DoCollision(VAR rp: RastPort);
(* Test every gel in gel list for collisions.
```

```
rp - a RastPort *)
```

```
PROCEDURE DrawGList(VAR rp: RastPort; VAR vp: ViewPort);
(* Process the gel list, queueing VSprites, drawing Bobs.
```

```
rp - the RastPort where Bobs will be drawn
vp - the ViewPort for which VSprites will be created *)
```

```
PROCEDURE FreeGBuffers(VAR anOb: AnimOb; VAR rp: RastPort; db: BOOLEAN);
(* Deallocate memory obtained by GetGBuffers.
```

```
anOb - the AnimOb structure
```

```

rp - the current RastPort
db - double-buffer indicator (set TRUE for double-buffering *)

PROCEDURE GetGBuffers(VAR anOb: AnimOb; VAR rp: RastPort; db: BOOLEAN): BOOLEAN;
(* Attempt to allocate ALL buffers of an entire AnimOb.

anOb - the AnimOb structure
rp - the current RastPort
db - double-buffer indicator (set TRUE for double-buffering)

result - TRUE if the memory allocations were all successful, else FALSE *)

PROCEDURE InitAnimation(VAR animKey: AnimObPtr);
(* Procedure implementation of a 'C' macro:
* #define InitAnimate(animKey) {*(animKey) = NULL;}

animKey - address of the variable that points to the head AnimOb *)

PROCEDURE InitGels(VAR head, tail: VSprite; VAR GInfo: GelsInfo);
(* Initialize a gel list; must be called before using gels.

head - the VSprite structure to be usedw as the gel list head
tail - the VSprite structure to be usedw as the gel list tail
GInfo - the GelsInfo structure to be initialized *)

PROCEDURE InitGMasks(VAR anOb: AnimOb);
(* Initialize all of the masks of an AnimOb.

anOb - the AnimOb structure *)

PROCEDURE InitMasks(VAR vs: VSprite);
(* Initialize the BorderList and CollMask masks of a VSprite.

vs - the VSprite structure *)

PROCEDURE RemBob(VAR bob: Bob);
(* Procedure implementation of a 'C' macro:
* #define RemBob(b) {(b)->Flags |= BOBSAWAY;}

bob - Bob structure *)

PROCEDURE RemIBob(VAR bob: Bob; VAR rp: RastPort; VAR vp: ViewPort);
(* Immediately remove a Bob from the gel list and the RastPort.

bob - the Bob to be removed
rp - the RastPort if the Bob is to be erased
vp - the ViewPort for beam-synchornizing *)

PROCEDURE RemVSprite(VAR vs: VSprite);
(* Remove a VSprite from the current gel list.

vs - the VSprite structure to be removed from the gel list *)

PROCEDURE SetCollision(num: LONGCARD; routine: ADDRESS; VAR GInfo: GelsInfo);
(* Set a pointer to a use collision routine.

num - collision vector number
routine - pointer to the user's collision routine
GInfo - a GelsInfo structure *)

PROCEDURE SortGList(VAR rp: RastPort);
(* Sort the current gel list, ordering it y,x coordinates.

rp - the RastPort structure containing the GelsInfo *)

END Gels.

```

Module Graphics

```
(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: Graphics.DEF      Version: Amiga.00.00
* Created: 11/20/86      Updated: 12/30/86      Author: Leon Frenkel
* Description: Graphics types/functions.
*****)
```

DEFINITION MODULE Graphics;

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)

CONST
 GraphicsName = "graphics.library";

TYPE
 RectanglePtr = POINTER TO Rectangle;
 Rectangle = RECORD
 MinX : INTEGER;
 MinY : INTEGER;
 MaxX : INTEGER;
 MaxY : INTEGER;
 END;

PointPtr = POINTER TO Point;
 Point = RECORD
 x : INTEGER;
 y : INTEGER;
 END;

PlanePtr = ADDRESS;

BitMapPtr = POINTER TO BitMap;
 BitMap = RECORD
 BytesPerRow : CARDINAL;
 Rows : CARDINAL;
 Flags : BYTE;
 Depth : BYTE;
 pad : CARDINAL;
 Planes : ARRAY [0..7] OF PlanePtr;
 END;

PROCEDURE InitBitMap(VAR bm: BitMap; depth, width, height: INTEGER);
(* Initialize bit map structure with input values.

bm - a BitMap structure
 depth - number of bitplanes that this bitmap will have
 width - number of bits (columns) wide for this BitMap
 height - number of bits (rows) tall for this BitMap *)

END Graphics.

```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: GraphicsBase.DEF      Version: Amiga.00.00
* Created: 11/29/86   Updated: 05/07/87   Author: Leon Frenkel
* Description: Graphics Library Base definitions.
*****)

```

```
DEFINITION MODULE GraphicsBase;
```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM Blit IMPORT bltnodeptr;
FROM Copper IMPORT copinitPtr;
FROM Interrupts IMPORT Interrupt;
FROM Libraries IMPORT Library;
FROM Lists IMPORT List;
FROM Semaphores IMPORT SignalSemaphorePtr;
FROM Tasks IMPORT TaskPtr;
FROM Text IMPORT TextFontPtr;
FROM Views IMPORT ViewPtr;

```

```
TYPE
```

```

GfxBasePtr = POINTER TO GfxBase;
GfxBase = RECORD
  LibNode      : Library;
  ActiView     : ViewPtr;
  copinit      : copinitPtr; (* ptr to copper start up list *)
  cia          : ADDRESS;    (* for 8520 resource use *)
  blitter      : ADDRESS;    (* for future blitter resource use *)
  LOFlist      : ADDRESS;
  SHFlist      : ADDRESS;
  blthd        : bltnodeptr;
  blttl        : bltnodeptr;
  bsblthd      : bltnodeptr;
  bsblttl      : bltnodeptr;
  vbsrv        : Interrupt;
  timsrv       : Interrupt;
  bltsrv       : Interrupt;
  TextFonts    : List;
  DefaultFont  : TextFontPtr;
  Modes        : CARDINAL; (* copy of current bplcon0 *)
  VBlank       : BYTE;
  Debug        : BYTE;
  BeamSync     : INTEGER;
  (* init'd to 0, it is ored into each bplcon0 for display *)
  systembplcon0 : INTEGER;
  SpriteReserved : BYTE;
  bytereserved  : BYTE;
  (* candidates for removal *)
  Flags        : CARDINAL;
  BlitLock     : INTEGER;
  BlitNest     : INTEGER;
  BlitWaitQ    : List;
  BlitOwner    : TaskPtr;
  TOFWaitQ     : List;
  (* NTSC PAL GENLOC etc. Display flags are determined at power on *)
  DisplayFlags : BITSET;
  SimpleSprites : ADDRESS;
  MaxDisplayRow : CARDINAL;
  MaxDisplayColumn : CARDINAL;
  NormalDisplayRows : CARDINAL;
  NormalDisplayColumns : CARDINAL;
  NormalDPMX    : CARDINAL; (* Dots per meter on display *)
  NormalDPHY    : CARDINAL; (* Dots per meter on display *)
  LastChanceMemory : SignalSemaphorePtr;
  LCMptr        : ADDRESS;
  MicrosPerLine : CARDINAL; (* 256 timer usec/line *)
  reserved      : ARRAY [0..1] OF LONGCARD;
END;

```

CONST

NTSC = 0:

```
GENLOC = 1:
```

PAL = 2:

BlitMsgFault = 4:

```
END GraphicsBase.
```

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Heap.DEF                      Version: Amiga.00.00      *
* Created: 07/06/87   Updated: 07/06/87   Author: Leon Frenkel *
* Description: Memory management module as defined in appendix 2 of *
* "Programming in Modula-2" by N. Wirth. This module is included to *
* allow programs written on other systems to be ported and vice versa, *
* to access the full memory management capabilities of the Amiga use the *
* module "Memory". *
* This module may be used instead of Storage if it is necessary to be *
* able to deallocate all memory allocated so far quickly, by using the *
* FreeHeap() procedure. *
*****)

```

DEFINITION MODULE Heap;

FROM SYSTEM IMPORT ADDRESS;

(* \$L+*)

PROCEDURE ALLOCATE(VAR a: ADDRESS; size: LONGCARD);
 (* Allocate an area of given size and returns its address in a.

 a - variable to be set to pointer to memory, set to NIL if not enough mem
 size - number of bytes to allocate *)

PROCEDURE DEALLOCATE(VAR a: ADDRESS; size: LONGCARD);
 (* Frees the area at address a with the given size.

 a - variable which contains the adr of the memory to be deallocated
 size - number of bytes to free *)

PROCEDURE Available(size: LONGCARD): BOOLEAN;
 (* Available returns TRUE if size bytes could be allocated. Since the Amiga
 is a multi-tasking system, the only way to be sure that the memory will
 actually be available when the ALLOCATE is issued is to use Forbid() and
 Permit() around the call to this procedure and the call to the ALLOCATE
 procedure which may follow it.

 size - number of bytes we would require

 result - TRUE if the required amount of memory is available *)

PROCEDURE FreeHeap();
 (* Free all memory allocated with ALLOCATE() but not released with
 * DEALLOCATE().
 *)

END Heap.

Module InitMathLib0

```
(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: InitMathLib0.DEF      Version: Amiga.00.00
* Created: 06/29/87   Updated: 06/29/87   Author: Leon Frenkel
* Description: Provides for initialization of the module "MathLib0".
*****
*)

DEFINITION MODULE InitMathLib0;

(*$L+*)

PROCEDURE OpenMathLib0(): BOOLEAN;
(* Open the Amiga "mathtrans.library".

    result - TRUE = opened successfully, FALSE = failed to open *)

PROCEDURE CloseMathLib0();
(* Close the Amiga's "mathtrans.library". *)

END InitMathLib0.
```



```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: InOut.DEF                               Version: Amiga.00.00                               *
* Created: 11/19/86   Updated: 08/07/87   Author: Leon Frenkel                               *
* Description: Standard InOut module as described in appendix 2 of                               *
* Programming in Modula-2 by N. Wirth.                                                       *
*****)

DEFINITION MODULE InOut;
  FROM SYSTEM IMPORT WORD;
  FROM FileSystem IMPORT File;

  (*$L+*)

  CONST EOL = 12C;
  VAR Done: BOOLEAN;
      termCH: CHAR; (*terminating character in ReadInt, ReadCard*)
      in, out: File; (*for exceptional cases only*)

  (* TRUE = echo chars in ReadString, FALSE = don't echo *)
  Echo: BOOLEAN; (* Default: TRUE *)

  PROCEDURE OpenInput(defext: ARRAY OF CHAR);
    (*request a file name and open input file "in".
     Done := "file was successfully opened".
     If open, subsequent input is read from this file.
     If name ends with ".", append extension defext*)

  PROCEDURE OpenOutput(defext: ARRAY OF CHAR);
    (*request a file name and open output file "out"
     Done := "file was successfully opened".
     If open, subsequent output is written on this file*)

  PROCEDURE OpenInputFile(filename: ARRAY OF CHAR);
    (* open the file of the specified name.
     Done := "file was successfully opened".
     If open, subsequent input is read from this file*)

  PROCEDURE OpenOutputFile(filename: ARRAY OF CHAR);
    (* open the file of the specified name.
     Done := "file was successfully opened".
     If open, subsequent output is written on this file*)

  PROCEDURE CloseInput;
    (*closes input file; returns input to terminal*)

  PROCEDURE CloseOutput;
    (*closes output file; returns output to terminal*)

  PROCEDURE Read(VAR ch: CHAR);
    (*Done := NOT in.eof*)

  PROCEDURE ReadString(VAR s: ARRAY OF CHAR);
    (*read string, i.e. sequence of characters not containing
     blanks nor control characters; leading blanks are ignored.
     Input is terminated by any character <= " ";
     this character is assigned to termCH.
     DEL is used for backspacing when input from terminal*)

  PROCEDURE ReadInt(VAR x: INTEGER);
    (*read string and convert to integer. Syntax:
     integer = ["+"|"-"] digit {digit}.
     Leading blanks are ignored.
     Done := "integer was read"*)

  PROCEDURE ReadCard(VAR x: CARDINAL);
    (*read string and convert to cardinal. Syntax:
     cardinal = digit {digit}.
     Leading blanks are ignored.
     Done := "cardinal was read"*)

  PROCEDURE ReadWrd(VAR w: WORD);
    (*Done := NOT in.eof*)

```

Module InOut

```
PROCEDURE Write(ch: CHAR);
PROCEDURE WriteLn; (*terminate line*)
PROCEDURE WriteString(s: ARRAY OF CHAR);
PROCEDURE WriteInt(x: INTEGER; n: CARDINAL);
(*write integer x with (at least) n characters on file "out".
  If n is greater than the number of digits needed,
  blanks are added preceding the number*)
PROCEDURE WriteCard(x,n: CARDINAL);
PROCEDURE WriteOct(x,n: CARDINAL);
PROCEDURE WriteHex(x,n: CARDINAL);
PROCEDURE WriteWrd(w: WORD);
END InOut.
```

```

(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: InputDevice.DEF      Version: Amiga.00.00
* Created: 11/26/86   Updated: 11/26/86   Author: Leon Frenkel
* Description: "input.device" types.
*****)

```

```
DEFINITION MODULE InputDevice;
```

```
FROM IODevices IMPORT CmdNonStd;
```

```
CONST
```

```

  INDAddHandler   = CmdNonStd + 0;
  INDRemHandler   = CmdNonStd + 1;
  INDWriteEvent   = CmdNonStd + 2;
  INDSetThresh    = CmdNonStd + 3;
  INDSetPeriod    = CmdNonStd + 4;
  INDSetMPort     = CmdNonStd + 5;
  INDSetMType     = CmdNonStd + 6;
  INDSetMTrig     = CmdNonStd + 7;

```

```
END InputDevice.
```

Module InputEvents

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: InputEvents.DEF                      Version: Amiga.00.00      *
* Created: 11/26/86 Updated: 05/07/86 Author: Leon Frenkel          *
* Description: Amiga InputEvents types.                                     *
*****)
```

DEFINITION MODULE InputEvents;

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM TimerDevice IMPORT timeval;

TYPE

```
(* InputEvent.ieClass *)
IEClass = (IEClassNull, (* a NOP input event *)
            IEClassRawKey, (* a raw keycode from the keyboard device *)
            IEClassRawMouse, (* raw mouse report from the gameport dev*)
            IEClassEvent, (* a private console event *)
            IEClassPointerPos, (* a pointer position report *)
            IEClass5, (* not used *)
            IEClassTimer, (* a timer event *)
            IEClassGadgetDown, (* select button pressed down over gadget*)
            IEClassGadgetUp, (* select button released over same gadget*)
            IEClassRequester, (* some requester activity has taken place*)
            IEClassMenuList, (* this is a Menu Number transmission *)
            IEClassCloseWindow, (* active windows close gadget selected *)
            IEClassSizeWindow, (* the window is a new size *)
            IEClassRefreshWindow, (* window needs to be refreshed *)
            IEClassNewPrefs, (* new preferences are available *)
            IEClassDiskRemoved, (* disk has been removed *)
            IEClassDiskInserted, (* disk has been inserted *)
            IEClassActiveWindow, (* window is about to be made active *)
            IEClassInactiveWindow); (* window is about to be made inactive *)
```

CONST

IEClassMax = IEClassInactiveWindow; (* the last class *)

(* InputEvent.ieCode *)

(* IEClassRawKey *)

IECodeUpPrefix = 80H;
IECodeKeyCodeFirst = 00H;
IECodeKeyCodeLast = 77H;
IECodeCommCodeFirst = 78H;
IECodeCommCodeLast = 7FH;

(* IEClassANSI *)

IECodeCOFirst = 000H;
IECodeCOLast = 01FH;
IECodeASCIIFirst = 020H;
IECodeASCIIILast = 07EH;
IECodeASCIIIdel = 07FH;
IECodeC1First = 080H;
IECodeC1Last = 09FH;
IECodeLatin1First = 0A0H;
IECodeLatin1Last = 0FFH;

(* IEClassRawMouse *)

IECodeLButton = 068H; (* also uses IECodeUpPrefix *)
IECodeRButton = 069H;
IECodeMButton = 06AH;
IECodeNoButton = 0FFH;

(* IEClassEvent *)

IECodeNewActive = 01H; (* active input window changed *)

(* IEClassRequester Codes *)

IECodeReqSet = 01H; (* first Requester opens in window *)
IECodeReqClear = 00H; (* last Requester clears out of the window *)

TYPE

```
(* InputEvent.ieQualifier *)
IEQualifier = (IEQualifierLShift,
IEQualifierRShift,
IEQualifierCapsLock,
IEQualifierControl,
IEQualifierLAlt,
IEQualifierRAlt,
IEQualifierLCommand,
IEQualifierRCommand,
IEQualifierNumericPad,
IEQualifierRepeat,
IEQualifierInterrupt,
IEQualifierMultiBroadcast,
IEQualifierMidButton,
IEQualifierRButton,
IEQualifierLeftButton,
IEQualifierRelativeMouse);
IEQualifierSet = SET OF IEQualifier;
```

(* InputEvent *)

InputEventPtr = POINTER TO InputEvent;

InputEvent = RECORD

```
ieNextEvent : InputEventPtr; (* the chronologically next event *)
ieClass : IEClass; (* the input event class *)
ieSubClass : BYTE; (* optional subclass of class *)
ieCode : CARDINAL; (* the input event code *)
ieQualifier : IEQualifierSet; (* qualifiers in effect *)
CASE : CARDINAL OF
| 0: ieX : INTEGER; (* pointer position for event *)
| ieY : INTEGER;
| 1: ieEventAddress : ADDRESS;
END;
ieTimeStamp : timeval; (* system tick at the event *)
END;
```

END InputEvents.

Module Interrupts

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Interrupts.DEF                      Version: Amiga.00.00 *
* Created: 11/19/86   Updated: 07/24/87   Author: Leon Frenkel *
* Description: Exec Interrupts Types/Functions. *
*****)
```

DEFINITION MODULE Interrupts;

FROM SYSTEM IMPORT ADDRESS;
FROM Lists IMPORT List;
FROM Nodes IMPORT Node, NodePtr;

TYPE

InterruptPtr = POINTER TO Interrupt;
Interrupt = RECORD
 isNode : Node;
 isData : ADDRESS; (* server data segment *)
 isCode : ADDRESS; (* server code entry *)
END;

(* For EXEC use ONLY! *)

IntVectorPtr = POINTER TO IntVector;

IntVector = RECORD
 ivData : ADDRESS;
 ivCode : ADDRESS;
 ivNode : NodePtr;
END;

(* For EXEC use ONLY! *)

SoftIntListPtr = POINTER TO SoftIntList;

SoftIntList = RECORD
 shList : List;
 shPad : CARDINAL;
END;

CONST

SIHPriMask = 00FOH;

(* This is a fake INT definition, use only for AddIntServer and the like *)

IntNMI = 15;

PROCEDURE AddIntServer(intNum: CARDINAL; VAR interrupt: Interrupt);

(* Add an interrupt server to the system.

intNum - the Portia interrupt bit number (0..14). Processor level seven
interrupts (NMI) are encoded as intNum 15
interrupt - an interrupt server node *)

PROCEDURE Cause(VAR interrupt: IntVector);

(* Cause a software interrupt.

interrupt - a properly initialized interrupt node *)

PROCEDURE Disable;

(* Disable interrupt processing. *)

PROCEDURE Enable;

(* Permit system interrupts to resume. *)

PROCEDURE Forbid;

(* Forbid task rescheduling. *)

PROCEDURE Permit;

(* Permit task rescheduling. *)

```
PROCEDURE RemIntServer(intNum: CARDINAL; VAR interrupt: Interrupt);
(* Remove an interrupt server.
```

```
    intNum - the Portia interrupt bit (0..14)
    interrupt - an interrupt server node *)
```

```
PROCEDURE SetIntVector(intNumber: CARDINAL; VAR interrupt: Interrupt): InterruptPtr;
(* Set a system interrupt vector.
```

```
    intNum - the Portia interrupt bit number (0..14)
    interrupt - an interrupt node structure
```

```
    result - a pointer to the prior interrupt node which had control of this
             interrupt *)
```

```
PROCEDURE SuperState(): ADDRESS;
(* Enter supervisor state with user stack.
```

```
    result - system stack pointer. Save this. It will come in useful when
             you return to user state. If the system is already in supervisor
             mode, result is zero. *)
```

```
PROCEDURE UserState(sysStack: ADDRESS);
(* Return to user state with user stack.
```

```
    sysStack - supervisor stack pointer *)
```

```
END Interrupts.
```

Module INTHardware

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: INTHardware.DEF                      Version: Amiga.00.00      *
* Created: 11/29/86   Updated: 11/30/86   Author: Leon Frenkel        *
* Description: Amiga INT control definitions.                          *
*****)
```

DEFINITION MODULE INTHardware;

(* \$L+ *)

CONST

```
(* Set/Clear control bit. Determines if bits written with a 1 get *)
(* set or cleared. Bits written with a zero are always unchanged *)
INTSetClr = 15;
```

```
INTInten    = 14; (* Master interrupt (enable only) *)
INTExter    = 13; (* External interrupt *)
INTDskSync  = 12; (* Disk re-SYNChronized *)
INTRBF      = 11; (* serial port Recieve Buffer Full *)
INTAud3     = 10; (* Audio channel 3 block finished *)
INTAud2     = 9;  (* Audio channel 2 block finished *)
INTAud1     = 8;  (* Audio channel 1 block finished *)
INTAud0     = 7;  (* Audio channel 0 block finished *)
INTBlit     = 6;  (* Blitter finished *)
INTVertB    = 5;  (* Start of Vertical Blank *)
INTCoper    = 4;  (* Coprocessor *)
INTPorts    = 3;  (* I/O Ports and timers *)
INTSoftInt  = 2;  (* software interrupt request *)
INTDskBlk   = 1;  (* Disk Block done *)
INTTBE      = 0;  (* serial port Transmit Buffer Empty *)
```

PROCEDURE OnVBlank;

(* Implementation of 'C' macro:

```
* #define ON_VBLANK      custom.intena = BITSET|INTF_VERTB
*)
```

PROCEDURE OffVBlank;

(* Implementation of 'C' macro:

```
* #define OFF_VBLANK     custom.intena = BITCLR|INTF_VERTB
*)
```

END INTHardware.


```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: Intuition.DEF           Version: Amiga.00.00                                       *
* Created: 11/24/86   Updated: 07/24/87   Author: Leon Frenkel                             *
* Description: Amiga Intuition types/functions.                                           *
*****)

```

```

DEFINITION MODULE Intuition;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT IntuitionBase; (* Used by IMPLEMENTATION MODULE *)
FROM Clipping IMPORT LayerPtr; IMPORT Clipping;
FROM Graphics IMPORT BitMapPtr; IMPORT Graphics;
FROM InputEvents IMPORT IEQualifier, IEQualifierSet, IECodeUpPrefix,
    IECodeLButton, IECodeRButton;
FROM KeyMapResource IMPORT KeyMapPtr;
FROM Memory IMPORT MemReqSet;
FROM Ports IMPORT MsgPortPtr, Message;
FROM Rasters IMPORT RastPort, RastPortPtr, Jam2, DrawModeSet; IMPORT Rasters;
FROM Text IMPORT TextAttrPtr, TextFontPtr;
FROM Views IMPORT ViewModesSet, ViewPortPtr, ViewPtr; IMPORT Views;

```

```

CONST
    IntuitionName = "intuition.library";

```

```

TYPE

```

```

    (* Forward Pointer Types *)

```

```

    MenuPtr      = POINTER TO Menu;
    MenuItemPtr  = POINTER TO MenuItem;
    RequesterPtr = POINTER TO Requester;
    GadgetPtr     = POINTER TO Gadget;
    IntuiTextPtr  = POINTER TO IntuiText;
    BorderPtr     = POINTER TO Border;
    ImagePtr      = POINTER TO Image;
    WindowPtr     = POINTER TO Window;

```

```

    (* ===== Menu ===== *)

```

```

    MenuFlags = (* Flags set by both the appliprogram and intuition *)
        (MenuEnabled, (* whether or not this menu is enabled *))

```

```

        MF1, MF2, MF3, MF4, MF5, MF6, MF7,

```

```

        (* Flags set by intuition *)

```

```

        MidDrawn; (* this menu's items are currently drawn *)

```

```

    MenuFlagsSet = SET OF MenuFlags;

```

```

    Menu = RECORD

```

```

        NextMenu : MenuPtr; (* same level *)
        LeftEdge : INTEGER; (* position of the select box *)
        TopEdge : INTEGER;
        Width : INTEGER; (* dimensions of the select box *)
        Height : INTEGER;
        Flags : MenuFlagsSet; (* see flag definitions above *)
        MenuName : ADDRESS; (* text for this Menu Header *)
        FirstItem : MenuItemPtr; (* pointer to first in chain *)

```

```

        (* these mysteriously-named variables are for internal use only *)

```

```

        JazzX : INTEGER;
        JazzY : INTEGER;
        BeatX : INTEGER;
        BeatY : INTEGER;

```

```

    END;

```

```

    (* ===== MenuItem ===== *)

```

```

    MenuItemFlags = (* flags set by the appliprogram *)

```

```

        (CheckIt, (* whether to check this item if selected *)
         ItemText, (* set if textual, clear if graphics item *)
         CommSeq, (* set if there's a command sequence *)
         MenuToggle, (* set to toggle the check of a menu item *)
         ItemEnabled, (* set if this item is enabled *)
         MIF5,
         MIF6, (* bit 6,7 are the special highlight flags *)
         MIF7,

```

Module Intuition

```

    Checked,      (* if CheckIt, then set this when selected *)
    MIF9,
    MIF10,
    MIF11,
    IsDrawn,      (* this item's subs are currently drawn *)
    HighItem,     (* this item is currently highlighted *)
    MenuToggled,  (* this item was already toggled *)
    MIF15);
MenuItemFlagsSet = SET OF MenuItemFlags;

CONST
(* these are the SPECIAL HIGHLIGHT FLAG state meanings *)
HighFlags = MenuItemFlagsSet{MIF6, MIF7}; (* see definitions below for these bits *)
HighImage = MenuItemFlagsSet{};           (* use the user's "select image" *)
HighComp = MenuItemFlagsSet{MIF6};        (* highlight by complementing the selectbox *)
HighBox = MenuItemFlagsSet{MIF7};        (* highlight by "boxing" the selectbox *)
HighNone = MenuItemFlagsSet{MIF6, MIF7}; (* don't highlight *)

TYPE
(* menu item mutual exclude bits *)
MenuItemMutualExcludeSet = SET OF [0..31];

MenuItem = RECORD
    NextItem      : MenuItemPtr; (* pointer to next in chained list *)
    LeftEdge      : INTEGER;      (* position of the select box *)
    TopEdge       : INTEGER;
    Width         : INTEGER;      (* dimensions of the select box *)
    Height        : INTEGER;
    Flags         : MenuItemFlagsSet; (* see the defines above *)

    (* set bits mean this item excludes that *)
    MutualExclude : MenuItemMutualExcludeSet;

    ItemFill      : ADDRESS; (* points to Image, IntuiText, or NIL *)

    (* when this item is pointed to by the cursor and the items
    * highlight mode HIGHIMAGE is selected, this alternate image
    * will be displayed
    *)
    SelectFill     : ADDRESS; (* points to Image, IntuiText, or NIL *)

    Command       : BYTE; (* only if appliprogram sets the CommSeq flag *)
    SubItem       : MenuItemPtr; (* if non-zero, DrawMenu shows "->*" *)

    (* The nextSelect field represents the menu number of next
    * selected item (when user has drag-selected several items)
    *)
    NextSelect    : CARDINAL;
END;

CONST
(* menu stuff *)
NoMenu = 0001FH;
NoItem = 0003FH;
NoSub = 0001FH;
MenuNull = OFFFFH;

TYPE
(* ===== Requester ===== *)
RequesterFlags = (* flags set by the appliprogram *)
    (PointRel,      (* if PointRel set, TopLeft is relative to pointer *)
    PreDrawn,      (* if ReqBMap points to predrawn Requester imagery *)
    NoisyReq,      (* if you don't want requester to filter input *)

    RF3, RF4, RF5, RF6, RF7, RF8, RF9, RF10, RF11,

    (* flags set by intuition *)
    ReqOffWindow,  (* part of one of the Gadgets was offwindow *)
    ReqActive,     (* this requester is active *)
    SysRequest,    (* this requester caused by system *)
    DeferRefresh); (* this requester stops a Refresh broadcast *)
RequesterFlagsSet = SET OF RequesterFlags;

Requester = RECORD
    (* the ClipRect and BitMap are used for rendering the requester *)

```

```

OlderRequest : RequesterPtr;
LeftEdge     : INTEGER;  (* dimensions of the entire box *)
TopEdge      : INTEGER;
Width        : INTEGER;  (* dimensions of the entire box *)
Height       : INTEGER;
RelLeft      : INTEGER;  (* for Pointer relativity offsets *)
RelTop       : INTEGER;

ReqGadget    : GadgetPtr; (* pointer to a list of Gadgets *)
ReqBorder    : BorderPtr; (* the box's border *)
ReqText      : IntuiTextPtr; (* the box's text *)
Flags        : RequesterFlagsSet; (* see definitions above *)

(* pen number for back-plane fill before draws *)
BackFill     : BYTE;
(* layer in place of clip rect *)
ReqLayer     : LayerPtr;

ReqPad1      : ARRAY [0..31] OF BYTE;

(* If the BitMap plane pointers are non-zero, this tells the system
 * that the image comes pre-drawn (if the appliprogram wants to define
 * its own box, in any shape or size it wants!); this is OK by
 * Intuition as long as there's a good correspondence between
 * the image and the specified Gadgets
 *)
(* points to the BitMap of PreDrawn imagery added *)
ImageBMap    : BitMapPtr;

RWindow      : WindowPtr; (* points back to Window *)

ReqPad2      : ARRAY [0..35] OF BYTE;
END;

(* ===== Gadget ===== *)
GadgetFlags = (* flags set by the appliprogram *)
  (GFO,          (* gadget highlight bits *)
   GF1,

  (* set this flag if the GadgetRender and SelectRender point to
   * Image imagery, clear if it's a Border
   *)
   GadgImage,

  (* combinations in these next two bits specify to which corner
   * the gadget's Left & Top coordinates are relative. If
   * relative to Top/Left, these are "normal" coordinates
   * (everything is relative to something in this universe)
   *)
   GRelBottom,   (* set if rel to bottom, clear if rel top *)
   GRelRight,    (* set if rel to right, clear if to left *)

  (* set the RelWidth bit to spec that Width is relative to width
   * of screen
   *)
   GRelWidth,

  (* set the RelHeight bit to spec that Height is rel to height
   * of screen
   *)
   GRelHeight,

  (* the Selected flag is initialized by you and set by Intuition
   * It specifies whether or not this Gadget is currently
   * selected/highlighted
   *)
   Selected,

  (* the GadgDisabled flag is initialized by you and later set by
   * Intuition according to your call to On/OffGadget(). It
   * specifies whether or not this Gadget is currently disabled
   * from begin selected
   *)
   GadgDisabled);
GadgetFlagsSet = SET OF GadgetFlags;

```

Module Intuition

CONST

```
(* these bits describe the highlight technique to be used *)
GadgHighBits = GadgetFlagsSet{GFO, GF1}; (* the highlight bits *)
GadgHComp    = GadgetFlagsSet{};          (* Complement the select box *)
GadgHBox     = GadgetFlagsSet{GFO};       (* Draw a box around the image *)
GadgHImage   = GadgetFlagsSet{GF1};       (* Blast in this alternate image *)
GadgHNone    = GadgetFlagsSet{GFO, GF1}; (* don't highlight *)
```

TYPE

```
(* These are the Activation flag bits *)
GadgetActivation = (* RelVerify is set if you want to verify that the pointer
    * was still over the gadget when the select button was
    * released
    *)
    (RelVerify,

    (* the flag GadgImmediate, when set, informs the caller
    * that the gadget was activated when it was activated
    * this flag works in conjunction with the RelVerify flag
    *)
    GadgImmediate,

    (* the flag EndGadget, when set, tells the system that
    * this gadget, when selected, causes the Requester
    * or AbsMessage to be ended. Requesters or AbsMessages
    * that are ended are erased and unlinked from the system
    *)
    EndGadget,

    (* the FOLLOWMOUSE flag, when set, specifies that you want to receive
    * reports on mouse movements (ie, you want the REPORTMOUSE function for
    * your Window). When the Gadget is deselected (immediately if you have
    * no RELVERIFY) the previous state of the REPORTMOUSE flag is restored
    * You probably want to set the GADGIMMEDIATE flag when using FOLLOWMOUSE,
    * since that's the only reasonable way you have of learning why Intuition
    * is suddenly sending you a stream of mouse movement events. If you don't
    * set RELVERIFY, you'll get at least one Mouse Position event.
    *)
    FollowMouse,

    (* if any of the BORDER flags are set in a Gadget that's
    * included in the Gadget list when a Window is opened,
    * the corresponding Border will be adjusted to make
    * room for the Gadget
    *)
    RightBorder,
    LeftBorder,
    TopBorder,
    BottomBorder,

    ToggleSelect, (* this bit for toggle-select mode *)

    StringCenter, (* should be a StringInfo flag, but it's ok*)
    StringRight,  (* should be a StringInfo flag, but it's ok*)

    LongInt,      (* this String Gadget is actually LONG Int *)

    AltKeyMap,    (* this String has an alternate keymap *)

    BoolExtend); (* this Boolean Gadget has a BoolInfo *)
GadgetActivationSet = SET OF GadgetActivation;
```

```
(* Gadget mutual exclusion bits *)
GadgetMutualExcludeSet = SET OF [0..31];
```

```
(* Gadget type set, bit descriptions follow *)
GadgetTypeSet = SET OF [0..15];
```

CONST

```
(* GADGET TYPES *)
(* These are the Gadget Type definitions for the variable GadgetType
    * gadget number type MUST start from one. NO TYPES OF ZERO ALLOWED.
    * first comes the mask for Gadget flags reserved for Gadget typing
    *)
GadgetType = GadgetTypeSet{10..15}; (* all Gadget Global Type flags (padded) *)
```

```

SysGadget = GadgetTypeSet{15}; (* 1 = SysGadget, 0 = AppliGadget *)
ScrGadget = GadgetTypeSet{14}; (* 1 = ScreenGadget, 0 = WindowGadget *)
GzzGadget = GadgetTypeSet{13}; (* 1 = Gadget for GimmeZeroZero borders *)
ReqGadget = GadgetTypeSet{12}; (* 1 = this is a Requester Gadget *)
(* system gadgets *)
Sizing = GadgetTypeSet{4};
WDragging = GadgetTypeSet{5};
SDragging = GadgetTypeSet{4,5};
WUpFront = GadgetTypeSet{6};
SUpFront = GadgetTypeSet{4,6};
WDownBack = GadgetTypeSet{5,6};
SDownBack = GadgetTypeSet{4,5,6};
Close = GadgetTypeSet{7};
(* application gadgets *)
BoolGadget = GadgetTypeSet{0};
GadgetOOO2 = GadgetTypeSet{1};
PropGadget = GadgetTypeSet{0,1};
StrGadget = GadgetTypeSet{2};

```

TYPE

```

Gadget = RECORD
    NextGadget : GadgetPtr; (* next gadget in the list *)
    LeftEdge : INTEGER; (* "hit box" of gadget *)
    TopEdge : INTEGER;
    Width : INTEGER; (* "hit box" of gadget *)
    Height : INTEGER;
    Flags : GadgetFlagsSet; (* see above for defines *)
    Activation : GadgetActivationSet; (* see above for defines *)
    GadgetType : GadgetTypeSet; (* see above for defines *)

    (* appliprogram can specify that the Gadget be rendered as either as
    * Border or an Image. This variable points to which (or equals
    * NIL if there's nothing to be rendered about this Gadget)
    *)
    GadgetRender : ADDRESS;

    (* appliprogram can specify "highlighted" imagery rather than
    * algorithmic this can point to either Border or Image data
    *)
    SelectRender : ADDRESS;

    GadgetText : IntuiTextPtr; (* text for this gadget *)

    (* by using the MutualExclude word, the appliprogram can describe
    * which gadgets mutually-exclude which other ones. The bits
    * in MutualExclude correspond to the gadgets in object containing
    * the gadget list. If this gadget is selected and a bit is set
    * in this gadget's MutualExclude and the gadget corresponding to
    * that bit is currently selected (e.g. bit 2 set and gadget 2
    * is currently selected) that gadget must be unselected.
    * Intuition does the visual unselecting (with checkmarks) and
    * leaves it up to the program to unselect internally
    *)
    MutualExclude : GadgetMutualExcludeSet;

    (* pointer to a structure of special data required by Proportional,
    * String and Integer Gadgets
    *)
    SpecialInfo : ADDRESS;

    GadgetID : CARDINAL; (* user-definable ID field *)

    UserData : ADDRESS; (* ptr to general purpose user data *)
END;

(* ===== BoolInfo ===== *)
(* This is the special data needed by an Extended Boolean Gadget
* Typically this structure will be pointed to by the Gadget field SpecialInfo
*)
BoolInfoFlags =
    (* set BoolInfo.Flags to this flag bit.
    * in the future, additional bits might mean more stuff hanging
    * off of BoolInfo.Reserved.
    *)
    (BoolMask, (* extension is for masked gadget *))

```


Module Intuition

```
(* Reserved flags *)
BIF1, BIF2, BIF3, BIF4, BIF5, BIF6, BIF7, BIF8,
BIF9, BIF10, BIF11, BIF12, BIF13, BIF14, BIF15);
BoolInfoFlagsSet = SET OF BoolInfoFlags;

BoolInfoPtr = POINTER TO BoolInfo;
BoolInfo = RECORD
    Flags      : BoolInfoFlagsSet; (* defined above *)

    (* bit mask for highlighting and selecting mask must follow
    * same rules as an Image plane. It's width and height are
    * determined by the width and height of the gadget's select
    * box. (i.e. Gadget.Width and .Height).
    *)
    Mask       : ADDRESS;

    Reserved   : LONGCARD; (* set to 0 *)
END;

(* ===== PropInfo ===== *)
(* this is the special data required by the proportional Gadget
* typically, this data will be pointed to by the Gadget variable SpecialInfo
*)
(* FLAG BITS *)
PropInfoFlags = (AutoKnob,      (* this flag sez: gimme that old auto-knob*)
FreeHoriz,    (* if set, the knob can move horizontally *)
FreeVert,     (* if set, the knob can move vertically *)
PropBorderless, (* if set, no border will be rendered *)

    (* reserved flags *)
    PIF4, PIF5, PIF6, PIF7,

    KnobHit,      (* set when this Knob is hit *)

    (* reserved flags *)
    PIF9, PIF10, PIF11, PIF12, PIF13, PIF14, PIF15);
PropInfoFlagsSet = SET OF PropInfoFlags;

PropInfoPtr = POINTER TO PropInfo;
PropInfo = RECORD
    Flags      : PropInfoFlagsSet; (* general purpose flag bits *)

    (* You initialize the Pot variables before the Gadget is added to
    * the system. Then you can look here for the current settings
    * any time, even while User is playing with this Gadget. To
    * adjust these after the Gadget is added to the System, use
    * ModifyProp(); The Pots are the actual proportional settings,
    * where a value of zero means zero and a value of MAXPOT means
    * that the Gadget is set to its maximum setting.
    *)
    (* 16-bit FixedPoint horizontal quantity percentage *)
    HorizPot   : CARDINAL;
    (* 16-bit FixedPoint vertical quantity percentage *)
    VertPot    : CARDINAL;

    (* the 16-bit FixedPoint Body variables describe what percentage of
    * the entire body of stuff referred to by this Gadget is actually
    * shown at one time. This is used with the AUTOKNOB routines,
    * to adjust the size of the AUTOKNOB according to how much of
    * the data can be seen. This is also used to decide how far
    * to advance the Pots when User hits the Container of the Gadget.
    * For instance, if you were controlling the display of a 5-line
    * Window of text with this Gadget, and there was a total of 15
    * lines that could be displayed, you would set the VertBody value to
    * (MAXBODY / (TotalLines / DisplayLines)) = MAXBODY / 3.
    * Therefore, the AUTOKNOB would fill 1/3 of the container, and
    * if User hits the Container outside of the knob, the pot would
    * advance 1/3 (plus or minus) If there's no body to show, or
    * the total amount of displayable info is less than the display area,
    * set the Body variables to the MAX. To adjust these after the
    * Gadget is added to the System, use ModifyProp();
    *)
    HorizBody  : CARDINAL; (* horizontal Body *)
    VertBody   : CARDINAL; (* vertical Body *)
```

```

(* these are the variables that Intuition sets and maintains *)
CWidth   : CARDINAL; (* Container width (absolute) *)
CHeight  : CARDINAL; (* Container height (absolute) *)
HPotRes  : CARDINAL; (* pot increments *)
VPotRes  : CARDINAL;
LeftBorder : CARDINAL; (* Container borders *)
TopBorder : CARDINAL;
END;

CONST
KnobHMin = 6;      (* minimum horizontal size of the Knob *)
KnobVMin = 4;      (* minimum vertical size of the Knob *)
MaxBody  = 0FFFFH; (* maximum body value *)
MaxPot   = 0FFFFH; (* maximum pot value *)

TYPE
(* ===== StringInfo ===== *)
(* this is the special data required by the string Gadget
 * typically, this data will be pointed to by the Gadget variable SpecialInfo
 *)
StringInfoPtr = POINTER TO StringInfo;
StringInfo = RECORD
    (* you init these var, and then Intuition maintains them *)
    Buffer      : ADDRESS; (* buffer containing the start/final string *)
    UndoBuffer : ADDRESS; (* optional buffer for undoing current entry *)
    BufferPos   : INTEGER; (* character position in Buffer *)
    MaxChars   : INTEGER; (* max # chars in Buffer (incl null) *)
    DispPos    : INTEGER; (* Buffer position of first displ char *)

    (* Intuition inits and maintaing these variables for you *)
    UndoPos    : INTEGER; (* char position in undo buffer *)
    NumChars   : INTEGER; (* # of chars currently in buffer *)
    DispCount  : INTEGER; (* # of whole chars visible in Container *)
    CLeft     : INTEGER; (* topleft offset of the container *)
    CTop      : INTEGER;
    LayerPtr   : Clipping.LayerPtr; (* the RastPort containing this Gadget *)

    (* you can initialize this variable before the gadget is
     * submitted to Intuition, and then examine it later to
     * discover what integer the user has entered (if the user
     * never plays with the gadget, the value will be unchanged
     * from your initial setting)
     *)
    LongInt    : LONGINT;

    (* If you want this Gadget to use your own Console
     * keymapping, you set the ALTKEYMAP bit in the Activation
     * flags of the Gadget, and then set this variable to point
     * to your keymap. If you don't set the AltKeyMap, you'll
     * get the standard ASCII keymapping.
     *)
    AltKeyMap  : KeyMapPtr;
END;

(* ===== IntuiText ===== *)
(* IntuiText is a series of strings that start with a screen location
 * (always relative to the upper-left corner of something) and then the
 * text of the string. The text is null-terminated.
 *)
IntuiText = RECORD
    FrontPen   : BYTE;      (* the pen numbers for rendering *)
    BackPen    : BYTE;
    DrawMode   : DrawModeSet; (* the mod for rendering the text *)
    LeftEdge   : INTEGER;   (* relative start location for text *)
    TopEdge    : INTEGER;   (* relative start location for text *)
    ITextFont  : TextAttrPtr; (* if NIL, you accept the default *)
    IText      : ADDRESS;   (* pointer to null-terminated text *)
    NextText   : IntuiTextPtr; (* cont. to TxWrite another text *)
END;

(* ===== Border ===== *)
(* Data type Border, used for drawing a series of lines which is intended for

```

Module Intuition

```
* use as a border drawing, but which may, in fact, be used to render any
* arbitrary vector shape.
* The routine DrawBorder sets up the RastPort with the appropriate
* variables, then does a Move to the first coordinate, then does Draws
* to the subsequent coordinates.
* After all the Draws are done, if NextBorder is non-zero we call DrawBorder
* recursively
*)
```

```
Border = RECORD
    LeftEdge   : INTEGER;    (* initial offsets from the origin *)
    TopEdge    : INTEGER;
    FrontPen   : BYTE;        (* pens numbers for rendering *)
    BackPen    : BYTE;
    DrawMode   : DrawModeSet; (* mode for rendering *)
    Count      : BYTE;        (* number of XY pairs *)
    XY         : ADDRESS;     (* vector coord pairs rel to LeftTop *)
    NextBorder : BorderPtr;   (* pointer to any other Border too *)
END;
```

```
(* ===== Image ===== *)
```

```
(* This is a brief image structure for very simple transfers of
* image data to a RastPort
*)
```

```
Image = RECORD
    LeftEdge   : INTEGER;    (* starting offset relative to some origin *)
    TopEdge    : INTEGER;    (* starting offset relative to some origin *)
    Width      : INTEGER;    (* pixel size (though data is word-aligned) *)
    Height     : INTEGER;    (* pixel size (though data is word-aligned) *)
    Depth      : INTEGER;    (* pixel size (though data is word-aligned) *)
    ImageData  : ADDRESS;    (* pointer to the actual word-aligned bits *)
```

```
(* the PlanePick and PlaneOnOff variables work much the same way as the
* equivalent GELS Bob variables. It's a space-saving
* mechanism for image data. Rather than defining the image data
* for every plane of the RastPort, you need define data only
* for the planes that are not entirely zero or one. As you
* define your Imagery, you will often find that most of the planes
* ARE just as color selectors. For instance, if you're designing
* a two-color Gadget to use colors two and three, and the Gadget
* will reside in a five-plane display, bit plane zero of your
* imagery would be all ones, bit plane one would have data that
* describes the imagery, and bit planes two through four would be
* all zeroes. Using these flags allows you to avoid wasting all
* that memory in this way: first, you specify which planes you
* want your data to appear in using the PlanePick variable. For
* each bit set in the variable, the next "plane" of your image
* data is blitted to the display. For each bit clear in this
* variable, the corresponding bit in PlaneOnOff is examined.
* If that bit is clear, a "plane" of zeroes will be used.
* If the bit is set, ones will go out instead. So, for our example:
* Gadget.PlanePick = 0x02;
* Gadget.PlaneOnOff = 0x01;
* Note that this also allows for generic Gadgets, like the
* System Gadgets, which will work in any number of bit planes.
* Note also that if you want an Image that is only a filled
* rectangle, you can get this by setting PlanePick to zero
* (pick no planes of data) and set PlaneOnOff to describe the pen
* color of the rectangle.
*)
```

```
PlanePick : BYTE;
PlaneOnOff : BYTE;
```

```
(* if the NextImage variable is not NIL, Intuition presumes that
* it points to another Image structure with another Image to be
* rendered
*)
```

```
NextImage : ImagePtr;
END;
```

```
(* ===== IntuiMessage ===== *)
```

```
IDCMPFlags = (SizeVerify,
    NewSize,
    RefreshWindow,
    MouseButtons,
```



```

MouseMove,
GadgetDown,
GadgetUp,
ReqSet,
MenuPick,
(* Changed from "CloseWindow" to "Closewindow" to avoid
 * confusion with the "CloseWindow" procedure in module Windows.
 *)
Closewindow,
RawKey,
ReqVerify,
ReqClear,
MenuVerify,
NewPrefs,
DiskInserted,
DiskRemoved,
WBenchMessage,
ActiveWindow,
InactiveWindow,
DeltaMove,
VanillaKey,
IntuiTicks,
IF23, IF24, IF25, IF26, IF27, IF28, IF29, IF30,

(* the IDCMP Flags do not use this special bit, which is
 * cleared when Intuition sends its special message to the
 * Task, and set when Intuition gets its Message back from the
 * Task. Therefore, I can check here to find out fast whether
 * or not this Message is available for me to send
 *)
LonelyMessage);
IDCMPFlagsSet = SET OF IDCMPFlags;

CONST
(* IDCMP Codes *)
(* This group of codes is for the MenuVerify function *)
MenuHot      = 0001H; (* IntuiWants verification or MenuCancel *)
MenuCancel   = 0002H; (* HOT Reply of this cancels Menu operation *)
MenuWaiting  = 0003H; (* Intuition simply wants a ReplyMsg() ASAP *)

(* These are internal tokens to represent state of verification attempts
 * shown here as a clue.
 *)
OkOk        = MenuHot;    (* guy didn't care *)
OkAbort     = 0004H;      (* window rendered question moot *)
OkCancel    = MenuCancel; (* window sent cancel reply *)

(* This group of codes is for the WBenchMessage messages *)
WBenchOpen  = 0001H;
WBenchClose = 0002H;

TYPE
IntuiMessagePtr = POINTER TO IntuiMessage;
IntuiMessage = RECORD
    ExecMessage : Message;

    (* the Class bits correspond directly with the IDCMP Flags,
     * except for the special bit LONELYMESSAGE (defined above)
     *)
    Class       : IDCMPFlagsSet;

    (* the Code field is for special values like MENU number *)
    Code        : CARDINAL;

    (* the Qualifier field is a copy of the current InputEvent's
     * Qualifier
     *)
    Qualifier    : IEQualifierSet;

    (* IAddress contains particular addresses for Intuition
     * functions, like the pointer to the Gadget or the Screen
     *)
    IAddress     : ADDRESS;

    (* when getting mouse movement reports, any event you get

```

Module Intuition

```

    * will have the the mouse coordinates in these variables.
    * the coordinates are relative to the upper-left corner of
    * your Window (GIMMEZEROZERO notwithstanding)
    *)
MouseX      : INTEGER;
MouseY      : INTEGER;

(* the time values are copies of the current system clock
 * time. Micros are in units of microseconds, Seconds in
 * seconds.
 *)
Seconds     : LONGCARD;
Micros      : LONGCARD;

(* the IDCMPWindow variable will always have the address of
 * the Window of this IDCMP
 *)
IDCMPWindow : WindowPtr;

SpecialLink : IntuiMessagePtr; (* system-use variable *)
END;

(* ===== Screen ===== *)
ScreenFlags = (* Flags set by Intuition *)
(SFO, SF1, SF2, SF3, (* screen type described below *)

Showtitle, (* this gets set by a call to ShowTitle() *)
(* Changed from "ShowTitle" to "Showtitle" to
 * avoid name conflict with the function of
 * the same name *)

Beeping, (* set when Screen is beeping *)

CustomBitmap, (* if you are supplying your own BitMap *)
(* Changed from "CustomBitMap" to "CustomBitmap"
 * to avoid name conflict with the NewScreen
 * structure field "CustomBitMap". *)

ScreenBehind, (* if you want screen to open behind already
 * open screens
 *)

ScreenQuiet, (* if you do not want Intuition to render
 * into your screen (gadgets, title)
 *)

SF9, SF10, SF11, SF12, SF13, SF14, SF15); (* reserved *)
ScreenFlagsSet = SET OF ScreenFlags;

CONST
(* The ScreenType bits are reserved for describing various Screen types
 * available under Intuition.
 *)
ScreenType = ScreenFlagsSet{SFO..SF3}; (* all the screens types available*)
WBenchScreen = ScreenFlagsSet{SFO}; (* Ta Da! The Workbench *)
CustomScreen = ScreenFlagsSet{SFO, SF1, SF2, SF3}; (* for the special look *)

StdScreenHeight = -1; (* supply in NewScreen.Height *)

TYPE
ScreenPtr = POINTER TO Screen;
Screen = RECORD
    NextScreen : ScreenPtr; (* linked list of screens *)
    FirstWindow : WindowPtr; (* linked list Screen's Windows *)

    LeftEdge : INTEGER; (* parameters of the screen *)
    TopEdge : INTEGER;
    Width : INTEGER; (* parameters of the screen *)
    Height : INTEGER;

    MouseY : INTEGER; (* position relative to upper-left *)
    MouseX : INTEGER;

    Flags : ScreenFlagsSet; (* see definitions above *)

```

```

Title      : ADDRESS;  (* null-terminated Title text *)
DefaultTitle : ADDRESS; (* for Windows without ScreenTitle *)

(* Bar sizes for this Screen and all Window's in this Screen *)
BarHeight  : BYTE;
BarVBorder : BYTE;
BarHBorder : BYTE;
MenuVBorder : BYTE;
MenuHBorder : BYTE;
WBotTop    : BYTE;
WBotLeft   : BYTE;
WBotRight  : BYTE;
WBotBottom : BYTE;

Font       : TextAttrPtr; (* this screen's default font *)

(* the display data structures for this Screen *)
ViewPort   : Views.ViewPort; (* describing the Screen's display *)
RastPort   : Rasters.RastPort; (* describing Screen rendering *)
BitMap     : Graphics.BitMap; (* extra copy of RastPort BitMap *)
LayerInfo  : Clipping.LayerInfo; (* each screen gets a LayerInfo *)

(* You supply a linked-list of Gadgets for your Screen.
 * This list DOES NOT include system Gadgets. You get the standard
 * system Screen Gadgets by default
 *)
FirstGadget : GadgetPtr;

DetailPen   : BYTE; (* for bar/border/gadget rendering *)
BlockPen   : BYTE;

(* the following variable(s) are maintained by Intuition to
 * support the DisplayBeep() color flashing technique
 *)
SaveColorO : CARDINAL;

(* This layer is for the Screen and Menu bars *)
BarLayer   : LayerPtr;

ExtData    : ADDRESS;
UserData   : ADDRESS; (* general-purpose ptr to User data ext *)
END;

```

```

(* ===== NewScreen ===== *)
NewScreenPtr = POINTER TO NewScreen;
NewScreen = RECORD
    LeftEdge   : INTEGER; (* Screen dimensions *)
    TopEdge    : INTEGER;
    Width      : INTEGER;
    Height     : INTEGER;
    Depth      : INTEGER;

    DetailPen   : BYTE; (* for bar/border/gadget rendering *)
    BlockPen   : BYTE;

    ViewModes   : ViewModesSet; (* the Modes for the ViewPort *)
    Type        : ScreenFlagsSet; (* the screen type (see above) *)
    Font        : TextAttrPtr; (* new Screen's default text attr *)
    DefaultTitle : ADDRESS; (* the default title for this Screen *)
    Gadgets     : GadgetPtr; (* your own Gadgets for this Screen *)

    (* if you are opening a CustomScreen and already have a BitMap
     * that you want used for your Screen, you set the flags
     * CustomBitMap in the Type field and you set this variable
     * to point to your BitMap structure. The structure will be
     * copied into your Screen structure, after which you may
     * discard your own BitMap if you want
     *)
    CustomBitMap : BitMapPtr;
END;

```

Module Intuition

```
(* ===== Window ===== *)

WindowFlags = (* Flags requested (not directly set though) by the appliprog *)
(WindowSizing, (* include sizing system-gadget? *)
 WindowDrag,   (* include dragging system-gadget? *)
 WindowDepth,  (* include depth arrangement gadget? *)
 WindowClose,  (* include close-box system-gadget? *)

 SizeBRight,   (* size gadget uses right border *)
 SizeBBottom,  (* size gadget uses bottom border *)

 WF6,          (* refresh modes: WF6, WF7 define below *)
 WF7,

 BackDrop,     (* this is an ever-popular BackDrop window *)

 Reportmouse,  (* set this to hear about every mouse move *)
               (* "Reportmouse" to avoid conflict with *)
               (* "ReportMouse()" function. *)

 GimmeZeroZero, (* make extra border stuff *)

 Borderless,   (* set this to get a Window sans border *)

 Activate,     (* when Window opens, it's the Active one *)

 (* Flags set by Intuition *)
 WindowActive, (* this window is the active one *)
 InRequest,    (* this window is in request mode *)
 MenuState,    (* this window is active with its Menus on *)

 (* Other user flags *)
 RMBTrap,      (* Catch RightMouseButton events for your own *)
 NoCareRefresh, (* not to be bothered with Refresh *)

 (* Reserved flags *)
 WF18, WF19, WF20, WF21, WF22, WF23,

 (* Other Intuition Flags *)
 WindowRefresh, (* Window is currently refreshing *)
 WBenchWindow,  (* WorkBench tool ONLY Window *)
 WindowTicked,  (* only one timer tick at a time *)

 (* Reserved flags *)
 WF27, WF28, WF29, WF30, WF31);

WindowFlagsSet = SET OF WindowFlags;

CONST
(* bits of Flags unused yet *)
(* in the file intuition.h the mask is $FCFC0000, but it should be $F8FC0000*)
SuperUnused = WindowFlagsSet{WF18..WF23, WF27..WF31};

(* refresh modes *)
(* combinations of the RefreshBits select the refresh type *)
RefreshBits = WindowFlagsSet{WF6, WF7};
SmartRefresh = WindowFlagsSet{};
SimpleRefresh = WindowFlagsSet{WF6};
SuperBitMap = WindowFlagsSet{WF7};
OtherRefresh = WindowFlagsSet{WF6, WF7};

TYPE
Window = RECORD
    NextWindow : WindowPtr; (* for the linked list in a screen *)

    LeftEdge : INTEGER; (* screen dimensions of window *)
    TopEdge : INTEGER;
    Width : INTEGER; (* screen dimensions of window *)
    Height : INTEGER;

    MouseY : INTEGER; (* relative to upper-left of window *)
    MouseX : INTEGER;

    MinWidth : INTEGER; (* minimum sizes *)
    MinHeight : INTEGER;
```

```

MaxWidth      : CARDINAL;      (* maximum sizes *)
MaxHeight     : CARDINAL;

Flags         : WindowFlagsSet; (* see above for defines *)

MenuStrip     : MenuPtr;       (* the strip of menu headers *)

Title         : ADDRESS;       (* the title text for this window *)

FirstRequest  : RequesterPtr;  (* all active Requesters *)

DMRequest     : RequesterPtr;  (* double-click Requester *)

ReqCount      : INTEGER;      (* count of reqs blocking window *)

WScreen       : ScreenPtr;     (* this Window's screen *)
RPort         : RastPortPtr;   (* this Window's very own RastPort *)

(* the border variables describe the window border. If you specify
 * GimmeZeroZero when you open the window, then the upper-left of the
 * ClipRect for this window will be upper-left of the BitMap (with correct
 * offsets when in SuperBitMap mode; you MUST select GimmeZeroZero when
 * using SuperBitMap). If you don't specify ZeroZero, then you save
 * memory (no allocation of RastPort, Layer, ClipRect and associated
 * BitMaps), but you also must offset all your writes by BorderTop,
 * BorderLeft and do your own mini-clipping to prevent writing over the
 * system gadgets
 *)
BorderLeft    : BYTE;
BorderTop     : BYTE;
BorderRight   : BYTE;
BorderBottom  : BYTE;
BorderRPort   : RastPortPtr;

(* You supply a linked-list of Gadgets for your Window.
 * This list DOES NOT include system gadgets. You get the standard
 * window system gadgets by setting flag-bits in the variable Flags
 * (see the bit definitions above)
 *)
FirstGadget   : GadgetPtr;

(* these are for opening/closing the windows *)
Parent        : WindowPtr;
Descendant    : WindowPtr;

(* sprite data information for your own Pointer
 * set these AFTER you Open the Window by calling SetPointer()
 *)
Pointer       : ADDRESS;
PtrHeight     : BYTE;
PtrWidth      : BYTE;
XOffset       : BYTE;
YOffset       : BYTE;

(* the IDCMP Flags and User's and Intuition's Message Ports *)
IDCMPFlags    : IDCMPFlagsSet; (* User-selected flags *)
UserPort      : MsgPortPtr;
WindowPort    : MsgPortPtr;
MessageKey    : IntuiMessagePtr;

DetailPen     : BYTE;          (* for bar/border/gadget rendering *)
BlockPen      : BYTE;

(* the CheckMark is a pointer to the imagery that will be used when
 * rendering MenuItems of this Window that want to be checkmarked
 * if this is equal to NULL, you'll get the default imagery
 *)
CheckMark     : ImagePtr;

(* if non nil, Screen title when Window is active *)
ScreenTitle   : ADDRESS;

(* These variables have the mouse coordinates relative to the
 * inner-Window of GIMMEZEROZERO Windows. This is compared with the

```

Module Intuition

```

* MouseX and MouseY variables, which contain the mouse coordinates
* relative to the upper-left corner of the Window, GIMMEZEROZERO
* notwithstanding
*)
GZZMouseX    : INTEGER;
GZZMouseY    : INTEGER;

(* these variables contain the width and height of the
* inner-Window of GimmeZeroZero Windows
*)
GZZWidth     : INTEGER;
GZZHeight    : INTEGER;

ExtData      : ADDRESS;

(* general-purpose pointer to user data extension *)
UserData     : ADDRESS;

(* this pointer keeps a duplicate of what. Window.RPort->Layer
* is supposed to be pointing at
*)
WLayer       : LayerPtr;

(* need to keep track of the font that OpenWindow opened,
* in case user SetFont's into RastPort
*)
IFont        : TextFontPtr;
END;

(* ===== NewWindow ===== *)
NewWindowPtr = POINTER TO NewWindow;
NewWindow = RECORD
    LeftEdge   : INTEGER; (* screen dimensions of window *)
    TopEdge    : INTEGER;
    Width      : INTEGER; (* screen dimensions of window *)
    Height     : INTEGER;

    DetailPen   : BYTE;    (* for bar/border/gadget rendering *)
    BlockPen    : BYTE;

    IDCMPFlags  : IDCMPFlagsSet; (* user-selected IDCMP flags *)

    Flags       : WindowFlagsSet; (* see Window struct for defs *)

    (* You supply a linked-list of Gadgets for your Window.
    * This list DOES NOT include system Gadgets. You get the
    * standard system Window Gadgets by setting flag-bits in the
    * variable Flags (see the bit definitions under the Window
    * structure definition)
    *)
    FirstGadget : GadgetPtr;

    (* the CheckMark is a pointer to the imagery that will be used
    * when rendering MenuItems of this Window that want to be
    * checkmarked if this is equal to NIL, you'll get the default
    * imagery
    *)
    CheckMark   : ImagePtr;

    Title       : ADDRESS; (* the title text for this window *)

    (* the Screen pointer is used only if you've defined a
    * CustomScreen and want this Window to open in it. If so,
    * you pass the address of the Custom Screen structure in this
    * variable. Otherwise, this variable is ignored and doesn't
    * have to be initialized.
    *)
    Screen      : ScreenPtr;

    (* SuperBitMap Window? If so, put the address of your BitMap
    * structure in this variable. If not, this variable is
    * ignored and doesn't have to be initialized
    *)
    BitMap      : BitMapPtr;

```



```

(* the values describe the minimum and maximum sizes of your Windows.
 * these matter only if you've chosen the WindowSizing Gadget option,
 * which means that you want to let the User to change the size of
 * this Window. You describe the minimum and maximum sizes that the
 * Window can grow by setting these variables. You can initialize
 * any one these to zero, which will mean that you want to duplicate
 * the setting for that dimension (if MinWidth == 0, MinWidth will be
 * set to the opening Width of the Window).
 * You can change these settings later using SetWindowLimits().
 * If you haven't asked for a Sizing Gadget, you don't have to
 * initialize any of these variables.
 *)
MinWidth    : INTEGER; (* minimums *)
MinHeight   : INTEGER;
MaxWidth    : CARDINAL; (* maximums *)
MaxHeight   : CARDINAL;

(* the type variable describes the Screen in which you want
 * this Window to open. The type value can either be
 * CustomScreen or one of the system standard Screen Types such
 * as WBenchScreen. See the type definitions under the Screen
 * structure
 *)
Type        : ScreenFlagsSet;
END;

(* ===== Remember ===== *)
(* this structure is used for remembering what memory has been allocated to
 * date by a given routine, so that a premature abort or systematic exit
 * can deallocate memory cleanly, easily, and completely
 *)
RememberPtr = POINTER TO Remember;
Remember = RECORD
    NextRemember : RememberPtr;
    RememberSize : LONGCARD;
    Memory       : ADDRESS;
END;

CONST
(* these defines are for the COMMSEQ and CHECKIT menu stuff. IF CHECKIT,
 * I'll use a generic Width (for all resolutions) for the CheckMark.
 * IF COMMSEQ, likewise I'll use this generic stuff
 *)
CheckWidth    = 19;
CommWidth     = 27;
LowCheckWidth = 13;
LowCommWidth  = 16;

(* these are the AlertNumber defines. if you are calling DisplayAlert()
 * the AlertNumber you supply must have the ALERT_TYPE bits set to one
 * of these patterns
 *)
AlertType     = 80000000H;
RecoveryAlert = 00000000H; (* the system can recover from this *)
DeadEndAlert  = 80000000H; (* no recovery possible, this is it *)

(* When you're defining IntuiText for the Positive and Negative Gadgets
 * created by a call to AutoRequest(), these defines will get you
 * reasonable-looking text. The only field without a define is the IText
 * field; you decide what text goes with the Gadget
 *)
AutoFrontPen  = BYTE(0);
AutoBackPen   = BYTE(1);
AutoDrawMode  = Jam2;
AutoLeftEdge  = 6;
AutoTopEdge   = 3;
AutoITextFont = NIL;
AutoNextText  = NIL;

(* RawMouse and Qualifiers (Console OR IDCMP) *)
SelectUp      = CARDINAL(BITSET(IECodeLButton) + BITSET(IECodeUpPrefix));
SelectDown    = IECodeLButton;
MenuUp        = CARDINAL(BITSET(IECodeRButton) + BITSET(IECodeUpPrefix));

```

Module Intuition

```
MenuDown = IECodeRButton;
AltLeft = IEQualifierSet{IEQualifierLAlt};
AltRight = IEQualifierSet{IEQualifierRAlt};
AmigaLeft = IEQualifierSet{IEQualifierLCommand};
AmigaRight = IEQualifierSet{IEQualifierRCommand};
AmigaKeys = AmigaLeft + AmigaRight;
```

(* Misc *)

```
CursorUp = 4CH;
CursorLeft = 4FH;
CursorRight = 4EH;
CursorDown = 4DH;
KeyCodeQ = 10H;
KeyCodeX = 32H;
KeyCodeN = 36H;
KeyCodeM = 37H;
KeyCodeV = 34H;
KeyCodeB = 35H;
```

(* ===== Utility Functions ===== *)

```
PROCEDURE AllocRemember(VAR RememberKey: RememberPtr; Size: LONGCARD;
                        Flags: MemReqSet): ADDRESS;
```

(* AllocMem and create a link node to make FreeMem easy.

RememberKey - a pointer to struct Remember. Before call init ptr to NIL
Size - the size in byte of the memory allocation
Flags - the specifications for the memory allocation

result - ptr to a block of memory or NIL if memory not allocated *)

```
PROCEDURE CurrentTime(Seconds, Micros: ADDRESS);
(* Get the current time values.
```

Seconds - pointer to a LONG variable to receive the current seconds value
Micros - pointer to a LONG variable to receive the current microseconds value
)

```
PROCEDURE DisplayAlert(AlertNumber: LONGCARD; String: ADDRESS;
                      Height: INTEGER): BOOLEAN;
```

(* Create the display of an Alert message.

AlertNumber - the number of this Alert Message. The only pertinent bits
of this number are the AlertType bit(s). The rest of the
number is ignored by this routine.

String - pointer to the Alert message string,
Height - minimum display lines required for your message

result - If this is a DEADEND_ALERT, FALSE is always the return value.
If this is a RECOVERY_ALERT. The return value will be TRUE if the
User presses the left mouse button in response to your message,
and FALSE if the User presses the right hand button in response to
your text, or if the alert could not be posted *)

```
PROCEDURE DoubleClick(StartSecs, StartMicros: LONGINT;
                     CurrentSecs, CurrentMicros: LONGINT): BOOLEAN;
```

(* Test two values for double click timing.

StartSecs - start of double-click time stamp seconds
StartMicros - start of double-click time stamp microseconds
CurrentSecs - end of double-click time stamp seconds
CurrentMicros - end of double-click time stamp microseconds

result - If the difference between the supplied timestamp values is within
the double-click time range in the current set of Preferences, this
function returns TRUE, else it returns FALSE *)

```
PROCEDURE DrawBorder(VAR rastPort: RastPort; VAR border: Border;
                   LeftOffset, TopOffset: INTEGER);
```

(* Draws the specified Border into the RastPort.


```

rastPort - the RastPort to receive the border rendering
border - a Border structure
LeftOffset - the offset which be added to each vector's x coordinate
TopOffset - the offset which be added to each vector's y coordinate *)

PROCEDURE DrawImage(VAR rastPort: RastPort; VAR image: Image;
                    LeftOffset, TopOffset: INTEGER);
(* Draws the specified Image into the RastPort.

rastPort - the RastPort to receive the image rendering
image - an Image structure
LeftOffset - the offset which be added to each image's x coordinate
TopOffset - the offset which be added to each image's y coordinate *)

PROCEDURE FreeRemember(VAR RememberKey: RememberPtr; ReallyForget: BOOLEAN);
(* Free memory allocated by calls to AllocRemember().

RememberKey - a pointer to struct Remember.
ReallyForget - FALSE = free only the link nodes, TRUE = free mem and nodes *)

PROCEDURE IntuiTextLength(VAR IText: IntuiText): CARDINAL;
(* Returns the length (pixel width) of an IntuiText.

IText - an instance of an IntuiText structure

result - returns the pixel-width of text specified by IntuiText data *)

PROCEDURE PrintIText(VAR rastPort: RastPort; VAR IText: IntuiText;
                    LeftOffset, TopOffset: INTEGER);
(* Prints the text according to the IntuiText argument.

rastPort - the RastPort destination of the text
IText - an instance of the structure IntuiText
LeftOffset - left offset of the IntuitText into the RastPort
TopOffset - top offset of the IntuitText into the RastPort *)

(* ===== Screen Functions ===== *)

PROCEDURE CloseScreen(VAR screen: Screen);
(* Closes an Intuition Screen.

screen - the Screen to be closed *)

PROCEDURE CloseWorkBench(): BOOLEAN;
(* Closes the Workbench Screen.

result - TRUE if workbench screen was closed successfully, FALSE if workbench
        was not open, or if it has windows open which are not WB drawers *)

PROCEDURE DisplayBeep(screen: ScreenPtr);
(* Flashes the video display.

screen - pointer to a Screen or NIL to beep every screen *)

PROCEDURE GetScreenData(Buffer: ADDRESS; Size: CARDINAL; Type: ScreenFlagsSet;
                       VAR screen: Screen): BOOLEAN;
(* Get copy of a screen data structure.

Buffer - pointer to a buffer into which data can be copied
Size - the size of the buffer provided, in bytes
Type - the screen type, such as (WBenchScreen,CustomScreen)
screen - ignored, unless type is CustomScreen, which results only in copying
        'size' bytes from 'screen' to 'buffer'

result - TRUE if successful, FALSE if standard screen of Type 'type' could
        not be opened. *)

```

Module Intuition

```
PROCEDURE MakeScreen(VAR screen: Screen);
```

```
(* Do an Intuition-integrated MakeVPort() of a custom screen.
```

```
screen - Custom Screen structure *)
```

```
PROCEDURE MoveScreen(VAR screen: Screen; DeltaX, DeltaY: INTEGER);
```

```
(* Attempts to move the Screen by increments provided.
```

```
screen - Screen structure
```

```
DeltaX - amount to move the screen on the x-axis, should be set to 0
```

```
DeltaY - amount to move the screen on the y-axis *)
```

```
PROCEDURE OpenScreen(VAR newScreen: NewScreen): ScreenPtr;
```

```
(* Open an Intuition Screen.
```

```
newScreen - instance of a NewScreen structure
```

```
result - if all is well, returns the pointer to your new Screen, if anything  
goes wrong, return NIL *)
```

```
PROCEDURE OpenWorkBench(): ScreenPtr;
```

```
(* Opens the WorkBench Screen.
```

```
result - if WorkBench Screen opened successfully or wasq already opened then  
returns ptr to Screen, else returns NIL if anything went wrong *)
```

```
PROCEDURE RemakeDisplay;
```

```
(* Remake the entire Intuition display, it does the equivalent of MakeScreen()  
for every Screen. *)
```

```
PROCEDURE RethinkDisplay;
```

```
(* The grand manipulator of the entire Intuition display. *)
```

```
PROCEDURE ScreenToBack(VAR screen: Screen);
```

```
(* Send the specified screen to the back of the display.
```

```
screen - Screen structure *)
```

```
PROCEDURE ScreenToFront(VAR screen: Screen);
```

```
(* Brings the specified screen to the front of the display.
```

```
screen - a Screen structure *)
```

```
PROCEDURE ShowTitle(VAR screen: Screen; ShowIt: BOOLEAN);
```

```
(* Set the Screen title bar display mode.
```

```
screen - a Screen structure
```

```
ShowIt - TRUE, FALSE describing whether to show or hide the scr title bar *)
```

```
PROCEDURE WBenchToBack(): BOOLEAN;
```

```
(* Sends the WorkBench Screen in back of all Screens.
```

```
result - TRUE if the WorkBench screen was opened, otherwise FALSE *)
```

```
PROCEDURE WBenchToFront(): BOOLEAN;
```

```
(* Brings the WorkBench Screen in front of all Screens.
```

```
result - TRUE if the WorkBench screen was opened, otherwise FALSE *)
```

```
(* ===== Window Functions ===== *)
```

```
PROCEDURE ActivateWindow(VAR window: Window);
```

```
(* Activate an Intuition Window.
```

```

window - a Window structure *)

PROCEDURE BeginRefresh(VAR window: Window);
(* Sets up a Window for optimized refreshing.

    window - Window structure which needs refreshing *)

PROCEDURE ClearDMRequest(VAR window: Window): BOOLEAN;
(* Clears (detaches) the DMRequest of the Window.

    window - Window from which the DMRequest is to be cleared

    result - if the DMRequest was not currently in use, zeroes out the DMRequest
              pointer in the Window and returns TRUE. If the DMRequest was in use
              doesn't change the pointer and returns FALSE *)

PROCEDURE ClearPointer(VAR window: Window);
(* Clears the Mouse Pointer definition from a Window.

    window - Window to be cleared of its Pointer definition *)

PROCEDURE CloseWindow(VAR window: Window);
(* Closes an Intuition Window.

    window - a Window structure *)

PROCEDURE EndRefresh(VAR window: Window; Complete: BOOLEAN);
(* Ends the optimized refresh state of the Window.

    window - Window currently in optimized refresh mode
    Complete - TRUE or FALSE describing whether or not this Window is completely
              refreshed *)

PROCEDURE ModifyIDCMP(VAR window: Window; IDCMPFlags: IDCMPFlagsSet);
(* Modify the state of the Window's IDCMPFlags.

    window - the Window structure containing the IDCMP Ports
    IDCMPFlags - the flag bits describing the new desired state of the IDCMP *)

PROCEDURE MoveWindow(VAR window: Window; DeltaX, DeltaY: INTEGER);
(* Ask intuition to move a Window.

    window - structure of the Window to be moved
    DeltaX - signed value describing how far to move the Window on the x-axis
    DeltaY - signed value describing how far to move the Window on the y-axis *)

PROCEDURE OpenWindow(VAR newWindow: NewWindow): WindowPtr;
(* Opens an Intuition Window.

    newWindow - instance of a NewWindow structure

    result - if all is well, returns the pointer to your new Window, if anything
            goes wrong returns NIL *)

PROCEDURE RefreshWindowFrame(VAR window: Window);
(* Ask intuition to redraw your window border/gadgets.

    window - a Window structure *)

PROCEDURE ReportMouse(VAR window: Window; Boolean: BOOLEAN);
(* Tells Intuition to report mouse movement.

    window - Window structure associated with this request
    Boolean - TRUE or FALSE value specifying whether to report mouse moves *)

```

Module Intuition

PROCEDURE SetDMRequest(VAR window: Window; VAR DMRequester: Requester);
(* Sets the DMRequest of the Window.

 window - Window from which the DMRequest is to be set
 DMRequester - a Requester structure *)

PROCEDURE SetPointer(VAR window: Window; Pointer: ADDRESS;
 Height, Width: INTEGER; XOffset, YOffset: INTEGER);
(* Sets a Window with its own Pointer.

 window - Window to receive this Pointer definition
 Pointer - pointer to the data definition of a Sprite
 Height - the height of the Pointer
 Width - the width of the sprite (must be less than or equal to sixteen)
 XOffset - the offset for your sperite from the Pointer position
 YOffset - the offset for your sperite from the Pointer position *)

PROCEDURE SetWindowTitles(VAR window: Window; WindowTitle, ScreenTitle: ADDRESS);
(* Sets the Window's titles for both Window and Screen.

 window - your Window structure
 WindowTitle - pointer to null-term string or -1 or 0
 ScreenTitle - pointer to null-term string or -1 or 0 *)

PROCEDURE SizeWindow(VAR window: Window; DeltaX, DeltaY: INTEGER);
(* Ask Intuition to size a Window.

 window - the structure of the Window to be sized
 DeltaX - signed value describing how much to size the Window on the x-axis
 DeltaY - signed value describing how much to size the Window on the y-axis *)

PROCEDURE ViewAddress(): ViewPtr;
(* Returns the address of the Intuition View structure.

 result - the address of the Intuition View structure *)

PROCEDURE ViewPortAddress(VAR window: Window): ViewPortPtr;
(* Returns the address of a Window's ViewPort structure.

 window - Window for which you want the ViewPort address
 result - returns the address of the ViewPort for that window *)

PROCEDURE WindowLimits(VAR window: Window; MinWidth, MinHeight: INTEGER;
 MaxWidth, MaxHeight: CARDINAL): BOOLEAN;
(* Set the minimum and maximum limits of the Window.

 window - Window structure
 MinWidth - the new min width, 0 if no change
 MinHeight - the new min height, 0 if no change
 MaxWidth - the new max width, 0 if no change
 MaxHeight - the new max height, 0 if no change

 result - returns TRUE if everything ok. If any of the parameter were out of
 range returns FALSE. The correct parameters will be changed. *)

PROCEDURE WindowToBack(VAR window: Window);
(* Ask Intuition to send this window to the back.

 window - the structure of the window to be sent to the back *)

PROCEDURE WindowToFront(VAR window: Window);
(* Ask Intuition to bring this Window to the front.

 window - the structure of the window to be brought to front *)

(* ===== Requester Functions ===== *)

```
PROCEDURE AutoRequest(VAR window: Window; BodyText: IntuiTextPtr;
    PositiveText, NegativeText: IntuiTextPtr;
    PositiveFlags, NegativeFlags: IDCMPFlagsSet;
    Width, Height: INTEGER): BOOLEAN;
(* Automatically build and get response from a Requester.
```

window - a Window structure
 BodyText - an IntuiText structure or NIL
 PositiveText - an IntuiText structure or NIL
 NegativeText - an IntuiText structure or NIL
 PositiveFlags - flags for the IDCMP
 NegativeFlags - flags for the IDCMP
 Width - width of the requester for rendering
 Height - height of the requester for rendering

result - returns TRUE or FALSE. See Intuition Docs for details *)

```
PROCEDURE BuildSysRequest(VAR window: Window; BodyText: IntuiTextPtr;
    PositiveText, NegativeText: IntuiTextPtr;
    IDCMPFlags: IDCMPFlagsSet;
    Width, Height: INTEGER): ADDRESS;
(* Build and display a system Requester.
```

window - a window structure
 BodyText - an IntuiText structure or NIL
 PositiveText - an IntuiText structure or NIL
 NegativeText - an IntuiText structure or NIL
 IDCMPFlags - the IDCMP flags you want used for the init of the IDCMP of the
 Window containing this Requester
 Width - the width of the requester to be rendered
 Height - the height of the requester to be rendered

result - If the Requester was successfully rendered in a Window, the value
 returned by this procedure is a pointer to the Window in which the
 Requester was rendered. If, however, the Requester cannot be rendered
 in the Window, this routine will have called DisplayAlert() before
 returning and will pass back TRUE if the user pressed the left mouse
 button and FALSE if the user pressed the right mouse button.
 FALSE = \$00000000, TRUE = \$00000001 *)

```
PROCEDURE EndRequest(VAR requester: Requester; VAR window: Window);
(* Ends the Request and resets the Window.
```

requester - Requester to be removed
 window - window structure with which this Requester is associated with *)

```
PROCEDURE FreeSysRequest(VAR window: Window);
(* Frees resources used by a call to BuildSysRequest().
```

window - Window structure returned by a successful call to BuildSysRequest *)

```
PROCEDURE InitRequester(VAR requester: Requester);
(* Initialize a Requester structure.
```

requester - a Requester structure *)

```
PROCEDURE Request(VAR requester: Requester; VAR window: Window): BOOLEAN;
(* Activates a Requester.
```

requester - Requester to be displayed
 window - Window into which this Requester goes

result - TRUE if requester successfully opened, otherwise FALSE *)

(* ===== Menu Functions ===== *)

```
PROCEDURE ClearMenuStrip(VAR window: Window);
```

Module Intuition

(* Clears (detaches) the Menu strip from the Window.

window - a Window structure *)

PROCEDURE ItemAddress(VAR MenuStrip: Menu; MenuNumber: CARDINAL): MenuItemPtr;
(* Returns the address of the specified MenuItem.

MenuStrip - the first Menu in your menu strip

MenuNumber - the value which contains the packed data that selects the menu
and item and subitem.

result - If MenuNumber = MenuNull this routine returns NIL else returns
the address of the specified MenuNumber *)

PROCEDURE OffMenu(VAR window: Window; MenuNumber: CARDINAL);
(* Disable the given menu or menu item.

window - a window structure

MenuNumber - the menu piece to be disabled *)

PROCEDURE OnMenu(VAR window: Window; MenuNumber: CARDINAL);
(* Enables the given menu or menu item.

window - a Window structure

MenuNumber - the menu piece to be enabled *)

PROCEDURE SetMenuStrip(VAR window: Window; VAR menu: Menu);
(* Attaches the Menu strip to the Window.

window - a Window structure

menu - the first Menu in the Menu strip *)

(*****
(* The following procedures are implementations of usefull 'C' macros *)
(* for handling of menu numbers. *)
(***)

PROCEDURE MENUNUM(n: CARDINAL): CARDINAL;
(* Extract the ordinal menu number from the value.
* 'C' macro: #define MENUNUM(n) (n & 0x1F)

n - a packed menu code

result - extracted menu number *)

PROCEDURE ITEMNUM(n: CARDINAL): CARDINAL;
(* Extract the ordinal item number from value.
* #define ITEMNUM(n) ((n >> 5) & 0x003F)

n - a packed menu code

result - extracted item number *)

PROCEDURE SUBNUM(n: CARDINAL): CARDINAL;
(* Extract the ordinal sub-item number from the value.
* #define SUBNUM(n) ((n >> 11) & 0x001F)

n - a packed menu code

result - extracted sub-item number *)

(* ===== Gadget Functions ===== *)

PROCEDURE ActivateGadget(VAR gadget: Gadget; VAR window: Window;
Request: RequesterPtr): BOOLEAN;
(* Activate a (String) Gadget.

gadget - Gadget that you want activated
 window - a Window structure containing the Gadget
 Request - a Requester (may be NIL if this isn't a Requester Gadget)

result - TRUE If the conditions above are met, else FALSE *)

```
PROCEDURE AddGadget(VAR window: Window; VAR gadget: Gadget;
                   Position: CARDINAL): CARDINAL;
(* Add a Gadget to the Gadget list of the Window or Screen.
```

window - Window to get your gadget
 gadget - new Gadget
 Position - position in the list for the new Gadget (starting from zero as
 the first position in the list)

result - Returns the position of where the Gadget was actually added *)

```
PROCEDURE AddGList(VAR window: Window; VAR gadget: Gadget; Position: CARDINAL;
                  Numgad: CARDINAL; requester: RequesterPtr): CARDINAL;
(* Add a linked list of gadgets to a Window or Requester.
```

window - Window to get your Gadget
 gadget - the first Gadget to be added
 Position - position in the list for the new Gadget (starting from zero as
 the first position in the list)
 Numgad - the number of gadgets from the linked list to be added. \$FFFF means
 to add the entire null-terminated list of gadgets
 requester - the requester the gadgets will be added to if the ReqGadget flag
 is set for the first gadget in the list

result - returns the position of where the first gadget was actually added *)

```
PROCEDURE ModifyProp(VAR gadget: Gadget; VAR window: Window;
                    requester: RequesterPtr; Flags: PropInfoFlagsSet;
                    HorizPos, VertPot: CARDINAL; HorizBody, VertBody: CARDINAL);
(* Modify the current parameters of a Proportional Gadget.
```

gadget - a Proportional Gadget
 window - the window containing the gadget or the window containing the requester
 containing the gadget
 requester - a Requester (may be NIL if this isn't a requester gadget)
 Flags - value to be stored in the Flags variable of the PropInfo
 HorizPot - value to be stored in the HorizPot variable of the PropInfo
 VertPot - value to be stored in the VertPot variable of the PropInfo
 HorizBody - value to be stored in the HorizBody variable of the PropInfo
 VertBody - value to be stored in the VertBody variable of the PropInfo *)

```
PROCEDURE NewModifyProp(VAR gadget: Gadget; VAR window: Window;
                       requester: RequesterPtr; Flags: PropInfoFlagsSet;
                       HorizPot, VertPot: CARDINAL;
                       HorizBody, VertBody: CARDINAL; NumGad: INTEGER);
(* ModifyProp, but with Selective Refresh.
```

gadget - a Proportional Gadget
 window - the window containing the gadget or the window containing the requester
 containing the gadget
 requester - a Requester (may be NIL if this isn't a Requester Gadget)
 Flags - value to be stored in the Flags variable of the PropInfo
 HorizPot - value to be stored in the HorizPot variable of the PropInfo
 VertPot - value to be stored in the VertPot variable of the PropInfo
 HorizBody - value to be stored in the HorizBody variable of the PropInfo
 VertBody - value to be stored in the VertBody variable of the PropInfo
 NumGad - number of gadgets to be refresh after proppadget internals have
 been adjusted. \$FFFFFFFF (-1) means "to end of list." *)

```
PROCEDURE OffGadget(VAR gadget: Gadget; VAR window: Window;
                   requester: RequesterPtr);
(* Disables the specified Gadget.
```

gadget - Gadget that you want disabled

Module Intuition

window - a Window structure containing the Gadget or containing the requester
which contains the gadget
requester - a Requester (may be NIL if this isn't a Requester gadget *)

```
PROCEDURE OnGadget(VAR gadget: Gadget; VAR window: Window;  
    requester: RequesterPtr);  
(* Enables the specified gadget.
```

gadget - Gadget that you want enabled
window - a Window structure containing the Gadget or containing the requester
which contains the gadget
requester - a Requester (may be NIL if this isn't a Requester gadget *)

```
PROCEDURE RefreshGadgets(VAR gadgets: Gadget; VAR window: Window;  
    requester: RequesterPtr);  
(* Refresh (redraw) the Gadget display.
```

gadgets - the first in the list of Gadgets wanting refreshment
window - the Window containing the Gadget or its Requester
requester - a Requester (ignored if Gadget is not attached to a Requester) *)

```
PROCEDURE RefreshGList(VAR gadgets: Gadget; VAR window: Window;  
    requester: RequesterPtr; NumGad: INTEGER);  
(* Refresh (redraw) a chosen number of gadgets.
```

gadgets - first in the list of Gadgets wanting refreshment
window - the Window containing the Gadget or its Requester
requester - a Requester (ignored if Gadget is not attached to a Requester)
NumGad - maximum number of gadgets to be refreshed. A value of -1
will cause all gadgets to be refreshed from Gadget to the
end of the list. A value of -2 will also do this, but if Gadget
is a Requester Gadget (REQGADGET) ALL gadgets in the requester
will be refreshed (this is a mode compatible with v1.1
RefreshGadgets() *)

```
PROCEDURE RemoveGadget(VAR window: Window; VAR gadget: Gadget): INTEGER;  
(* Removes a Gadget from a Window.
```

window - the Window containing the Gadget or the Requester
gadget - the Gadget to be removed.
result - returns the ordinal position of the removed gadget. If the gadget
wasn't found in the appropriate list, or if there are no gadgets
in the list returns \$FFFF (-1) *)

```
PROCEDURE RemoveGList(VAR window: Window; VAR gadget: Gadget;  
    NumGad: INTEGER): INTEGER;  
(* Removes a sublist of Gadgets from a Window.
```

window - Window containing the Gadget or the Requester
gadget - the list of Gadgets to be removed.
NumGad - number of gadgets to be removed, or -1 remove to end of list
result - returns the ordinal position of the removed Gadget. If the
Gadget wasn't found in the appropriate list, or if there are no
Gadgets in the list returns \$FFFF (-1) *)

END Intuition.


```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                   *
*****
* Name: IntuitionBase.DEF                               Version: Amiga.00.00
* Created: 11/27/86   Updated: 05/21/87   Author: Leon Frenkel
* Description: Intuition Library Base and utility functions.
*****)

```

```
DEFINITION MODULE IntuitionBase;
```

```

IMPORT System; (* Used by IMPLEMENTATION MODULE *)
FROM SYSTEM IMPORT ADDRESS;
FROM InputEvents IMPORT InputEvent;
FROM Libraries IMPORT Library;
FROM Ports IMPORT MsgPort;
FROM Views IMPORT View;

```

```
TYPE
```

```

(* Be sure to protect yourself against someone modifying this data as
 * you look at it. This is done by calling:
 * lock := LockIBase(0). When done call UnlockIBase(lock).
 *)
(* IntuitionBase should never be directly modified by programs! *)
IntuitionBasePtr = POINTER TO IntuitionBase;
IntuitionBase = RECORD
    LibNode      : Library;

    ViewLord     : View;

    ActiveWindow : ADDRESS; (* WindowPtr *)
    ActiveScreen : ADDRESS; (* ScreenPtr *)

    (* The FirstScreen points to the frontmost Screen. Screens
     * are then maintained in a front to back order using
     * Screen.NextScreen
     *)
    FirstScreen  : ADDRESS; (* ScreenPtr *)

    Flags        : LONGCARD;

    MouseY       : INTEGER; (* mouse position relative to View*)
    MouseX       : INTEGER;
    Seconds      : LONGCARD; (* timestamp of most current *)
    Micros       : LONGCARD; (* input event *)

    (* Private part of Intuition Library data follows: *)
END;

```

```

PROCEDURE AlohaWorkbench(VAR WBPort: MsgPort);
(* Make presence and departure of WorkBench known to Intuition.

```

```

    WBPort - an initialize MsgPort structure where special communications
    are to take place *)

```

```

PROCEDURE Intuition(VAR InputEvent: InputEvent);
(* Main entry point into Intuition, where input events arrive and are dispatched

    InputEvents - the first input event in a linked list of InputEvent struc *)

```

```

PROCEDURE LockIBase(LockNumber: LONGCARD): LONGCARD;
(* Intuition user's access to Intuition Locking.

```

```

    LockNumber - Intuition lock desired. Should be 0 (zero).

```

```

    result - returns a value which should be passed to UnlockIBase() to release
    lock *)

```

Module IntuitionBase

```
PROCEDURE UnlockIBase(Lock: LONGCARD);
(* Surrender an Intuition lock gotten by LockIBase().
```

```
Lock - the value returned by LockIBase *)
```

```
END IntuitionBase.
```

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: IODevices.DEF                      Version: Amiga.00.00      *
* Created: 11/17/86   Updated: 05/07/87   Author: Leon Frenkel      *
* Description: Exec I/O Devices Types/Functions.                    *
*****)

```

```

DEFINITION MODULE IODevices;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE, LONGWORD;
FROM Libraries IMPORT Library;
FROM Ports IMPORT MsgPortPtr, Message;

```

```

TYPE

```

```

DevicePtr = POINTER TO Device;
Device = RECORD
  ddLibrary : Library;
END;

```

```

UnitFlags = (UnitActive, UnitInTask);
UnitFlagsSet = SET OF UnitFlags;

```

```

UnitPtr = POINTER TO Unit;
Unit = RECORD
  unitMsgPort : MsgPortPtr; (* queue for unprocessed messages *)
  unitFlags : UnitFlagsSet;
  unitpad : BYTE;
  unitOpenCnt : CARDINAL; (* number of active opens *)
END;

```

```

IOFlagsSet = SET OF [0..7];

```

```

IORequestPtr = POINTER TO IORequest;
IORequest = RECORD
  ioMessage : Message;
  ioDevice : DevicePtr; (* device node pointer *)
  ioUnit : UnitPtr; (* unit (driver private) *)
  ioCommand : CARDINAL; (* device command *)
  ioFlags : IOFlagsSet;
  ioError : BYTE; (* error or warning num *)
END;

```

```

IOStdReqPtr = POINTER TO IOStdReq;
IOStdReq = RECORD
  ioMessage : Message;
  ioDevice : DevicePtr; (* device node pointer *)
  ioUnit : UnitPtr; (* unit (driver private) *)
  ioCommand : CARDINAL; (* device command *)
  ioFlags : IOFlagsSet;
  ioError : BYTE; (* error or warning num *)
  ioActual : LONGCARD; (* actual number of bytes transferred *)
  ioLength : LONGCARD; (* requested number bytes transferred *)
  ioData : ADDRESS; (* points to data area *)
  ioOffset : LONGCARD; (* offset for block structured devices *)
END;

```

```

CONST

```

```

(* library vector offsets for device reserved vectors *)
DevBeginIO = -30;
DevAbortIO = -36;

```

```

(* ioFlags defined bits *)
IOQuick = 0;

```

```

(* ioCommand device commands *)
CmdInvalid = 0;
CmdReset = 1;
CmdRead = 2;
CmdWrite = 3;
CmdUpdate = 4;
CmdClear = 5;
CmdStop = 6;

```

Module IODevices

```
CmdStart   = 7;
CmdFlush   = 8;
CmdNonStd  = 9;
```

(* IO Error Codes *)

```
IOErrOpenFail = -1; (* device/unit failed to open *)
IOErrAborted  = -2; (* request aborted *)
IOErrNoCmd    = -3; (* command not supported *)
IOErrBadLength = -4; (* not a valid length *)
```

```
PROCEDURE AddDevice(VAR device: Device);
(* Add a device to the system.
```

device - a properly initialized device node *)

```
PROCEDURE CloseDevice(ioRequest: ADDRESS);
(* Conclude access to a device.
```

ioRequest - pointer to an I/O request structure *)

```
PROCEDURE OpenDevice(devName: ADDRESS; unitNumber: LONGINT;
ioRequest: ADDRESS; flags: LONGWORD): LONGINT;
```

(* Gain access to a device.

devName - requested device name
unitNumber - the unit number to open on that device. The formal of the unit number is device specific
ioRequest - pointer to the I/O request block to be returned with appropriate fields initialized
flags - additional driver specific information. This is sometimes used to request opening a device with excluding access

result - zero if successful, else an error is returned *)

```
PROCEDURE RemDevice(VAR device: Device): LONGINT;
(* Remove a device from the system.
```

device - a device node

result - zero if successful, else an error is returned *)

```
PROCEDURE AbortIO(ioRequest: ADDRESS): LONGINT;
(* Attempt to abort an in-progress I/O request.
```

ioRequest - pointer to an I/O request block

result - zero if successful, else an error is returned *)

```
PROCEDURE BeginIO(ioRequest: ADDRESS);
```

(* Initiate an I/O request. The request will be synchronous or asynchronous depending on the device driver.

ioRequest - pointer to an I/O request block *)

```
PROCEDURE CheckIO(ioRequest: ADDRESS): ADDRESS;
(* Get the IO request status.
```

ioRequest - pointer to an I/O request block

result - pointer to an IORequest or NIL if the I/O still in progress *)

```

PROCEDURE DoIO(ioRequest: ADDRESS): LONGINT;
(* Perform an I/O command and wait for completion.

```

```

    ioRequest - pointer to a properly initialized I/O request

```

```

    result - zero if successful, else an error is returned *)

```

```

PROCEDURE SendIO(ioRequest: ADDRESS);
(* Initiate an I/O command.

```

```

    ioRequest - pointer to an I/O request *)

```

```

PROCEDURE WaitIO(ioRequest: ADDRESS): LONGINT;
(* Wait for completion of an I/O request.

```

```

    ioRequest - pointer to an I/O request block

```

```

    result - zero if successful, else an error is returned *)

```

```

END IODevices.

```

Module IODevicesUtil

```
(*****
*                               Modula-2 Software Construction Set          *
*                               (c) 1986 by Leon Frenkel                     *
*****
* Name: IODevicesUtil.DEF          Version: Amiga.00.00                    *
* Created: 01/04/87   Updated: 01/16/87   Author: Leon Frenkel             *
* Description: Exec Support Procedures for Device I/O.                     *
*****)
```

DEFINITION MODULE IODevicesUtil;

FROM SYSTEM IMPORT ADDRESS;

FROM IODevices IMPORT IOStdReqPtr;

FROM Ports IMPORT MsgPort;

(*\$L+*)

PROCEDURE CreateExtIO(VAR ioReplyPort: MsgPort; size: LONGINT): ADDRESS;

(* Create an I/O request block of a user-specified number of bytes.

ioReplyPort - an already initialize message port to be used for this I/O
request's reply port.

size - the size of the I/O request to be created

result - Pointer to the new io request block or NIL if not enough memory
or the reply port was not a valid port *)

PROCEDURE CreateStdIO(VAR ioReplyPort: MsgPort): IOStdReqPtr;

(* Create a standard I/O request.

ioReplyPort - an already initialize message port to be used for this I/O
request's reply port.

result - Pointer to the new io request block or NIL if not enough memory
or the reply port was not a valid port *)

PROCEDURE DeleteExtIO(ioReq: ADDRESS);

(* Return memory allocated for an extended I/O request.

ioReq - pointer to the IORequest block to be freed *)

PROCEDURE DeleteStdIO(ioStdReq: IOStdReqPtr);

(* Return memory allocated for a standard I/O request.

ioReq - pointer to the IOStdReq block whose resources are to be freed *)

END IODevicesUtil.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: KeyboardDevice.DEF      Version: Amiga.00.00      *
* Created: 11/26/86   Updated: 11/26/86   Author: Leon Frenkel      *
* Description: "keyboard.device" types.      *
*****)

```

```
DEFINITION MODULE KeyboardDevice;
```

```
FROM IODevices IMPORT CmdNonStd;
```

```
CONST
```

```

KBDReadEvent      = CmdNonStd + 0;
KBDReadMatrix     = CmdNonStd + 1;
KBDAddResetHandler = CmdNonStd + 2;
KBDRemResetHandler = CmdNonStd + 3;
KBDResetHandlerDone = CmdNonStd + 4;

```

```
END KeyboardDevice.
```

Module KeyMapResource

```
(*****
*                               *
*      Modula-2 Software Construction Set                               *
*      (c) 1986 by Leon Frenkel                                         *
*                               *
*****
* Name: KeyMapResource.DEF      Version: Amiga.00.00                  *
* Created: 11/26/86   Updated: 11/26/86   Author: Leon Frenkel        *
* Description: "keymap.resource" and "console.device" keymap definitions *
*****)
```

DEFINITION MODULE KeyMapResource;

FROM SYSTEM IMPORT ADDRESS;
FROM Lists IMPORT List;
FROM Nodes IMPORT Node;

TYPE

KeyMapPtr = POINTER TO KeyMap;

KeyMap = RECORD

 kmLoKeyMapTypes : ADDRESS; (* ptr to BYTE *)
 kmLoKeyMap : ADDRESS; (* ptr to LONGCARD *)
 kmLoCapsable : ADDRESS; (* ptr to BYTE *)
 kmLoRepeatable : ADDRESS; (* ptr to BYTE *)
 kmHiKeyMapTypes : ADDRESS; (* ptr to BYTE *)
 kmHiKeyMap : ADDRESS; (* ptr to LONGCARD *)
 kmHiCapsable : ADDRESS; (* ptr to BYTE *)
 kmHiRepeatable : ADDRESS; (* ptr to BYTE *)

END;

KeyMapNodePtr = POINTER TO KeyMapNode;

KeyMapNode = RECORD

 knNode : Node; (* including name of keymap *)
 knKeyMap : KeyMap;

END;

(* the structure of keymap.resource *)

KeyMapResourcePtr = POINTER TO KeyMapResource;

KeyMapResource = RECORD

 krNode : Node;
 krList : List; (* a list of KeyMapNodes *)

END;

(* Key Map Types *)

KeyMapCode = (KCShift,
 KCAlt,
 KCControl,
 KCDownUp,
 KCDead,
 KCString,
 KCNoop);

KeyMapCodeSet = SET OF KeyMapCode;

CONST

KCNoQual = KeyMapCodeSet{};

KCVanilla = KeyMapCodeSet{KCShift, KCAlt, KCControl};

TYPE

(* Dead Prefix Bytes *)

DeadPrefix = (DPMOD,

 DP1,

 DP2,

 DPDead);

DeadPrefixSet = SET OF DeadPrefix;

CONST

(* mask for index for 1st of two dead keys *)

DP2DIndexMask = DeadPrefixSet{DPMOD..DPDead};

(* shift for factor for 1st of two dead keys *)

DP2DFacShift = 4;

END KeyMapResource.


```

(*****
*
*          Modula-2 Software Construction Set
*          (c) 1986 by Leon Frenkel
*
*****
* Name: LargeSets.DEF          Version: Amiga.00.00
* Created: 06/29/87   Updated: 06/29/87   Author: Leon Frenkel
* Description: Implements machine independant large sets (> machine word)
*   The set range for LargeSets is [0..65535]
* WARNING: Parameters to set functions are not checked for validity.
*   Set functions which take two sets as parameters require that both sets
*   have been created with the same range!
*****)

DEFINITION MODULE LargeSets;

FROM SYSTEM IMPORT ADDRESS;

(*$L+*)

TYPE
  LargeSet; (* LargeSet handle *)

PROCEDURE CreateSet(VAR set: LargeSet; first, last: CARDINAL): BOOLEAN;
(* Allocate memory for a LargeSet with the specified range. Set initially empty.

   set - variable to store handle for LargeSet
   first - first element in range of set
   last - last element in range of set

   result - TRUE = allocated successfully, FALSE = set allocation failed *)

PROCEDURE DeleteSet(set: LargeSet);
(* Deallocate a LargeSet allocated with CreateSet().

   set - handle of LargeSet to be deleted *)

PROCEDURE SetInfo(set: LargeSet; VAR setAdr: ADDRESS; VAR setSize: CARDINAL);
(* Return internal information about set, its address and size.

   set - handle of LargeSet
   setAdr - variable to store address of first byte of LargeSet
   setSize - variable to store size of LargeSet in bytes *)

PROCEDURE SetRange(set: LargeSet; VAR first, last: CARDINAL);
(* Return the range of a set, same as parameters passed to CreateSet().

   set - handle of LargeSet
   first - variable to store first element of set
   last - variable to store last element of set *)

PROCEDURE In(set: LargeSet; element: CARDINAL): BOOLEAN;
(* Test if element present in set.

   set - handle of LargeSet
   element - element to test

   result - TRUE = element in set, FALSE = element not in set *)

PROCEDURE IncludeElement(set: LargeSet; element: CARDINAL);
(* Include an element in set.

   set - handle of LargeSet
   element - element to include *)

```

Module LargeSets

```
PROCEDURE ExclElement(set: LargeSet; element: CARDINAL);
(* Exclude an element from set.
```

```
    set - handle of LargeSet
    element - element to exclude *)
```

```
PROCEDURE InclRange(set: LargeSet; first, last: CARDINAL);
(* Include range of elements in set.
```

```
    set - handle to LargeSet
    first - first element of range to be included
    last - last element of range to be included *)
```

```
PROCEDURE ExclRange(set: LargeSet; first, last: CARDINAL);
(* Exclude range of elements in set.
```

```
    set - handle to LargeSet
    first - first element of range to be excluded
    last - last element of range to be excluded *)
```

```
PROCEDURE CopySet(dstSet, srcSet: LargeSet);
(* Copy srcSet to dstSet.
```

```
    dstSet - destination for copy operation
    srcSet - source for copy operation *)
```

```
PROCEDURE Union(dstSet, srcSet: LargeSet);
(* Perform a union operation on two sets. Set operator: +
```

```
    dstSet - destination for union operation
    srcSet - source for union operation *)
```

```
PROCEDURE Diff(dstSet, srcSet: LargeSet);
(* Perform a difference operation on two sets. Set operator: -
```

```
    dstSet - destination for difference operation
    srcSet - source for difference operation *)
```

```
PROCEDURE Intersection(dstSet, srcSet: LargeSet);
(* Perform an intersection operation on two sets. Set operator: *
```

```
    dstSet - destination for intersection operation
    srcSet - source for intersection operation *)
```

```
PROCEDURE SymmetricDiff(dstSet, srcSet: LargeSet);
(* Perform a symmetric difference operation on two sets. Set operator: /
```

```
    dstSet - destination for symmetric difference operation
    srcSet - source for symmetric difference operation *)
```

```
END LargeSets.
```

```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: Layers.DEF                               Version: Amiga.00.00
* Created: 11/28/86   Updated: 07/24/87   Author: Leon Frenkel
* Description: "layers.library" functions.
*****)

```

```

DEFINITION MODULE Layers;

```

```

FROM System IMPORT LayersBase; (* used by IMPLEMENTATION MODULE *)
FROM Clipping IMPORT Layer, LayerPtr, LayerInfo, LayerInfoPtr,
  LayerInfoFlagsSet, ClipRect;
FROM Graphics IMPORT BitMap, BitMapPtr;
FROM Regions IMPORT Region, RegionPtr;
FROM Rasters IMPORT RastPort;

```

```

CONST

```

```

  LayersName = "layers.library";

```

```

PROCEDURE BeginUpdate(VAR l: Layer): BOOLEAN;
(* Prepare to repair damaged layer.

```

```

  l - a layer structure

```

```

  result - TRUE if damage list converted to ClipRect list successfully, FALSE
           if list conversion aborted (probably out of memory *)

```

```

PROCEDURE BehindLayer(VAR l: Layer): BOOLEAN;
(* Put layer behind other layers.

```

```

  l - a layer structure

```

```

  result - TRUE if operation successful or FALSE if unsuccessful *)

```

```

PROCEDURE CreateBehindLayer(VAR li: LayerInfo; VAR bm: BitMap; x0, y0: LONGINT;
  x1, y1: LONGINT; flags: LayerInfoFlagsSet;
  bm2: BitMapPtr): LayerPtr;
(* Create a new layer behind all existing layers.

```

```

  li - LayerInfo structure
  bm - common BitMap used by all Layers
  x0 - x component of upper left hand corner of layer
  y0 - y component of upper left hand corner of layer
  x1 - x component of lower right hand corner of layer
  y1 - y component of lower right hand corner of layer
  flags - various types of layers supported as bit sets.
  bm2 - optional Super BitMap ptr or NIL

```

```

  result - pointer to Layer structure if successful or NIL if not *)

```

```

PROCEDURE CreateUpfrontLayer(VAR li: LayerInfo; VAR bm: BitMap; x0, y0: LONGINT;
  x1, y1: LONGINT; flags: LayerInfoFlagsSet;
  bm2: BitMapPtr): LayerPtr;
(* Create a new layer on top of existing layers.

```

```

  li - LayerInfo structure
  bm - common BitMap used by all Layers
  x0 - x component of upper left hand corner of layer
  y0 - y component of upper left hand corner of layer
  x1 - x component of lower right hand corner of layer
  y1 - y component of lower right hand corner of layer
  flags - various types of layers supported as bit sets.
  bm2 - optional Super BitMap ptr or NIL

```

```

  result - pointer to Layer structure if successful or NIL if not *)

```

Module Layers

PROCEDURE DeleteLayer(VAR l: Layer): BOOLEAN;
(* Delete layer from layer list.

l - a layer structure

result - TRUE if layer deleted from the system, FALSE if layer not deleted*)

PROCEDURE DisposeLayerInfo(VAR li: LayerInfo);
(* Return all memory for LayerInfo to memory pool.

li - a LayerInfo structure *)

PROCEDURE EndUpdate(VAR l: Layer; flag: BOOLEAN);
(* Remove damage list and restore state of layer to normal.

l - a Layer structure

flag - use TRUE if update was completed. The damage list is cleared
use FALSE if update not complete. The damage list is retained *)

PROCEDURE InstallClipRegion(VAR l: Layer; VAR region: Region): RegionPtr;
(* Install clip region in layer.

l - a layer structure

region - a region

result - pointer to the previous ClipRegion that was installed or NIL
if no previous clip region *)

PROCEDURE LockLayer(VAR l: Layer);
(* Lock layer to make changes to ClipRects.

l - a Layer structure *)

PROCEDURE LockLayerInfo(VAR li: LayerInfo);
(* Lock the LayerInfo structure

li - a LayerInfo structure *)

PROCEDURE LockLayers(VAR li: LayerInfo);
(* Lock all layers from graphics output.

li - LayerInfo structure *)

PROCEDURE MoveLayer(VAR l: Layer; dx, dy: LONGINT): BOOLEAN;
(* Move layer to new position in BitMap.

l - nonbackdrop Layer structure

dx - delta to add to current x position

dy - delta to add to current y position

result - TRUE if operation successful, FALSE if failed (out of memory) *)

PROCEDURE MoveLayerInFrontOf(VAR layertomove, targetlayer: Layer): BOOLEAN;
(* Put layer in front of another layer.

layertomove - layer which should be moved

targetlayer - target layer in front of which to move layer

result - TRUE if operation successful, FALSE if failed (out of memory) *)

PROCEDURE NewLayerInfo(): LayerInfoPtr;
(* Allocate and Initialize full LayerInfo structure.

result - pointer to LayerInfo struc if successful or NIL if not enough mem *)

```

PROCEDURE ScrollLayer(VAR l: Layer; dx, dy: LONGINT);
(* Scroll around in a superbitmap, translate coordinates in non-superbitmap.

    l - a Layer structure
    dx - delta to add to current x scroll value
    dy - delta to add to current y scroll value *)

PROCEDURE SizeLayer(VAR l: Layer; dx, dy: LONGINT): BOOLEAN;
(* Change the size of this nonbackdrop layer.

    l - a nonbackdrop layer structure
    dx - delta to add to current x size
    dy - delta to add to current y size

    result - TRUE if operation successful, FALSE if failed (out of memory) *)

PROCEDURE SwapBitsRastPortClipRect(VAR rp: RastPort; VAR cr: ClipRect);
(* Swap bits between common bitmap and obscured ClipRect.

    rp - RastPort structure
    cr - ClipRect structure to swap bits with *)

PROCEDURE UnlockLayer(VAR l: Layer);
(* Unlock layer and allow graphics routines to use it.

    l - Layer structure *)

PROCEDURE UnlockLayerInfo(VAR li: LayerInfo);
(* Unlock the LayerInfo structure.

    li - LayerInfo structure *)

PROCEDURE UnlockLayers(VAR li: LayerInfo);
(* Unlock all layers from graphics output. Restart graphics output to layers
    that have been waiting.

    li - LayerInfo structure *)

PROCEDURE UpfrontLayer(VAR l: Layer): BOOLEAN;
(* Put layer in front of all other layers.

    l - a nonbackdrop layer structure

    result - TRUE if operation successful, FALSE if failed (out of memory) *)

PROCEDURE WhichLayer(VAR li: LayerInfo; x, y: LONGINT): LayerPtr;
(* Which layer is this point in?

    li - LayerInfo structure
    x - x component of coordinate
    y - y component of coordinate

    result - pointer to the topmost layer that this point is in, or NIL if this
    point is not in any layer *)

END Layers.

```

Module Libraries

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****)
* Name: Libraries.DEF      Version: Amiga.00.00
* Created: 11/15/86      Updated: 12/31/86      Author: Leon Frenkel
* Description: Exec Libraries Types/Functions.
*****)

DEFINITION MODULE Libraries;

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Nodes IMPORT Node;

CONST
  LibVectSize = 6; (* size of library vector entry *)
  LibReserved = 4; (* vectors reserved for special library functions *)
  LibBase = -LibVectSize; (* base adr of library *)
  LibUserDef = (LibBase - (LibReserved * LibVectSize));
  LibNonStd = LibUserDef; (* first user defined library function *)

  LibOpen = - 6; (* Open library vector *)
  LibClose = -12; (* Close library vector *)
  LibExpunge = -18; (* Expunge library vector *)
  LibExtFunc = -24; (* reserved library vector *)

TYPE
  LibFlags = (LibSumming, (* we are currently checksumming *)
              LibChanged, (* we have just changed the library *)
              LibSumUsed, (* set if we should bother to sum *)
              LibDelExp); (* delayed expunge *)

  LibFlagsSet = SET OF LibFlags;

  LibraryPtr = POINTER TO Library;
  Library = RECORD
    libNode      : Node;
    libFlags     : LibFlagsSet;
    libpad       : BYTE;
    libNegSize   : CARDINAL; (* number of bytes before library *)
    libPosSize   : CARDINAL; (* number of bytes after library *)
    libVersion   : CARDINAL;
    libRevision  : CARDINAL;
    libIdString  : ADDRESS;
    libSum       : LONGCARD; (* library checksum *)
    libOpenCnt   : CARDINAL; (* number of current opens *)
  END;

PROCEDURE AddLibrary(VAR library: Library);
(* Add a library to the system.

   library - a properly initialized library structure *)

PROCEDURE CloseLibrary(VAR library: Library);
(* Informs the system that the library will no longer be in used.

   library - a library node *)

PROCEDURE MakeFunctions(destination, functionArray, funcDispBase: ADDRESS): LONGCARD;
(* Construct a jump table of the type used by resources, libraries, device.

   destination - the target address for the function jump table
   functionArray - pointer to an array of function pointer or displacements
   funcDispBase - pointer to the base about which all function displacements
                  are relative. If zero then the functionArray contains abs ptrs

   result - size of the new table in bytes *)

```



```
PROCEDURE MakeLibrary(vectors, structure, init: ADDRESS; dataSize: LONGCARD;
                     segList: ADDRESS): LibraryPtr;
```

(* This function is used for constructing a library vector and data area.

vectors - pointer to an array of function pointer or function displacements.
If the first word of the array is -1, then the array contains
relative word displacements; otherwise, the array contains
absolute function pointers. The vector list is terminated by a
-1 (of the same size as the pointers).

structure - points to an "InitStruc" data region. If NIL then it will not
be called.

init - an entry point that will be called before adding the library to the
system. If NIL it will not be called. When called it will be passed
the libAddr in D0 and the segList parameter in A0. The result of
the init function will be the result returned by MakeLibrary.

dataSize - the size of the library data area, including the std library node

segList - ptr to a memory segment list (used by DOS)

result - the reference address of the library *)

```
PROCEDURE OpenLibrary(libName: ADDRESS; version: LONGCARD): LibraryPtr;
```

(* This function returns a pointer to a library previously installed in system.

libName - the name of the library to open

version - the version of the library required

result - a library pointer for a successful open, else NIL *)

```
PROCEDURE RemLibrary(VAR library: Library): LONGCARD;
```

(* Remove a library from the system.

library - pointer to a library node structure

result - zero if successful, else an error number *)

```
PROCEDURE SetFunction(VAR library: Library; funcOffset: INTEGER;
```

```
                     funcEntry: ADDRESS);
```

(* Change a function vector in a library.

library - the library to be changed

funcOffset - the offset that FuncEntry should be put at

funcEntry - pointer to new function

result - pointer to the old function which occupied the vector *)

```
PROCEDURE SumLibrary(VAR library: Library);
```

(* Compute and check the checksum on a library.

library - the library to be checksummed. If an old checksum does not
match and the library has not been marked as changed then the
system will alert the user. *)

END Libraries.

Module Lists

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Lists.DEF                      Version: Amiga.00.00      *
* Created: 11/15/86   Updated: 07/31/87   Author: Leon Frenkel   *
* Description: Exec List types/functions.                      *
*****)
```

DEFINITION MODULE Lists;

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM Nodes IMPORT Node, NodePtr, MinNodePtr;

TYPE

ListPtr = POINTER TO List;

List = RECORD

 lhHead : NodePtr; (* points to the first node in the list *)
 lhTail : NodePtr; (* is always NIL *)
 lhTailPred : NodePtr; (* points to the last node in the list *)
 lhType : BYTE; (* defines the type of nodes within the list *)
 lpad : BYTE; (* structure alignment byte *)

END;

MinListPtr = POINTER TO MinList;

MinList = RECORD

 mlhHead : MinNodePtr; (* points to the first node in the list *)
 mlhTail : MinNodePtr; (* is always NIL *)
 mlhTailPred : MinNodePtr; (* points to the last node in the list *)

END;

PROCEDURE AddHead(VAR list: List; VAR node: Node);
(* Add a node to the head of a doubly linked list.

 list - the target list header
 node - the node to insert at head *)

PROCEDURE AddTail(VAR list: List; VAR node: Node);
(* Add a node to the tail of a doubly linked list.

 list - the target list header
 node - the node to insert at tail *)

PROCEDURE Enqueue(VAR list: List; VAR node: Node);
(* Insert or append a node into a system que.

 list - the system que header
 node - the node to enqueue *)

PROCEDURE FindName(start: ADDRESS; name: ADDRESS): NodePtr;
(* Traverse a system list until a node with the given name is found

 start - a list header or list node to start the search at
 (if node this one is skipped)
 name - name string terminated with null

 result - a pointer to the node with the same name or NIL if not found *)

PROCEDURE Insert(VAR list: List; VAR node: Node; listNode: NodePtr);
(* Insert a node into a doubly linked list after a given node position.

 list - the target list header
 node - the node to insert
 listNode - the node after which to insert or NIL to insert at head *)


```

PROCEDURE NewList(VAR list: List);
(* Initialize the header of a list. Perform the following:
* list.lhHead := ADR(list.lhTail);
* list.lhTail := NIL;
* list.lhTailPred := ADR(list.lhHead);

list - the target list header *)

```

```

PROCEDURE RemHead(VAR list: List): NodePtr;
(* Get a pointer to the head node and remove it from the list.

list - the target list header

result - the node removed or NIL when empty list *)

```

```

PROCEDURE Remove(VAR node: Node);
(* Remove a node from a list.

node - the node to remove. *)

```

```

PROCEDURE RemTail(VAR list: List): NodePtr;
(* Get a pointer to the tail node and remove it from the list.

list - the target list header

result - the node removed or NIL when empty list *)

```

```

END Lists.

```

Module LongInOut

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: LongInOut.DEF      Version: Amiga.00.00 *
* Created: 07/22/87   Updated: 07/22/87   Author: Leon Frenkel *
* Description: Standard LongInOut module for LONGINT and LONGCARD types. *
*****)
```

DEFINITION MODULE LongInOut;

(* \$L+ *)

VAR Done: BOOLEAN; (* result of previous operation *)

```
PROCEDURE ReadLongInt(VAR x: LONGINT);
(*read string and convert to long integer. Syntax:
  long integer = ["+"|"-"] digit {digit}.
  Leading blanks are ignored.
  Done := "long integer was read"*)
```

```
PROCEDURE ReadLongCard(VAR x: LONGCARD);
(*read string and convert to long cardinal. Syntax:
  long cardinal = digit {digit}.
  Leading blanks are ignored.
  Done := "long cardinal was read"*)
```

```
PROCEDURE WriteLongInt(x: LONGINT; n: CARDINAL);
(*write long integer x with (at least) n characters on file "out".
  If n is greater than the number of digits needed,
  blanks are added preceding the number*)
```

```
PROCEDURE WriteLongCard(x: LONGCARD; n: CARDINAL);
PROCEDURE WriteLongOct(x: LONGCARD; n: CARDINAL);
PROCEDURE WriteLongHex(x: LONGCARD; n: CARDINAL);
```

END LongInOut.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: MathLib0.DEF              Version: Amiga.00.00      *
* Created: 12/24/86   Updated: 12/25/86   Author: Leon Frenkel *
* Description: This is the standard math module as defined in appendix 2 *
* of "Programming in Modula-2" by N. Wirth.                *
* WARNING: This module requires that the "mathfpf.library" has been opened *
* by the module "System". The "mathtrans.library" must also be *
* opened by your application in order to use this module.    *
*****)

```

```

DEFINITION MODULE MathLib0;

```

```

FROM System    IMPORT MathBase; (* used by IMPLEMENTATION MODULE *)
FROM Libraries IMPORT LibraryPtr;

```

```

CONST
  MathTransName = "mathtrans.library";

```

```

VAR
  MathTransBase : LibraryPtr;

```

```

CONST
  (* The following are often needed mathematical constants *)
  pi    = 3.14159265358979323846264338327950288;
  e     = 2.71828182845904523536028747135266250;
  sqrt2 = 1.41421356237309504880168872420969808;
  ln2   = 0.69314718055994530941723212145817657;

```

```

PROCEDURE arccos(x: REAL): REAL;
(* Obtain the arccosine of a number. *)

```

```

PROCEDURE arcsin(x: REAL): REAL;
(* Obtain the arcsine of a number. *)

```

```

PROCEDURE arctan(x: REAL): REAL;
(* Obtain the arctangent of a number. *)

```

```

PROCEDURE cos(x: REAL): REAL;
(* Obtain the cosine of a number. *)

```

```

PROCEDURE cosh(x: REAL): REAL;
(* Obtain the hyperbolic cosine of a number. *)

```

```

PROCEDURE DegToRad(x: REAL): REAL;
(* Convert degrees to radians. *)

```

```

PROCEDURE exp(x: REAL): REAL;
(* Obtain the exponent (e**x) of a number. *)

```

```

PROCEDURE ln(x: REAL): REAL;
(* Obtain the natural logarithm of a number. *)

```

```

PROCEDURE log(x: REAL): REAL;
(* Obtain the naparian logarithm (base 10) of a number. *)

```

```

PROCEDURE power(x, y: REAL): REAL;
(* Obtain the exponentiation of two numbers. x raised to y power. *)

```

Module MathLib0

```
PROCEDURE RadToDeg(x: REAL): REAL;  
(* Convert radians to degrees. *)
```

```
PROCEDURE sin(x: REAL): REAL;  
(* Obtain the sine of a number. *)
```

```
PROCEDURE sinh(x: REAL): REAL;  
(* Obtain the hyperbolic sine of a number. *)
```

```
PROCEDURE sqrt(x: REAL): REAL;  
(* Obtain the square root of a number. *)
```

```
PROCEDURE tan(x: REAL): REAL;  
(* Obtain the tangent of a number. *)
```

```
PROCEDURE tanh(x: REAL): REAL;  
(* Obtain the hyperbolic tangent of a number. *)
```

```
PROCEDURE real(x: LONGINT): REAL;  
(* Convert an integer into a real number. *)
```

```
PROCEDURE entier(x: REAL): LONGINT;  
(* Convert a real number into an integer. *)
```

```
END MathLib0.
```

```

*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: Memory.DEF               Version: Amiga.00.00                                     *
* Created: 11/17/86   Updated: 01/16/87   Author: Leon Frenkel                             *
* Description: Exec Memory Types/Functions.                                                 *
*****

```

```

DEFINITION MODULE Memory;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM Nodes IMPORT Node;

```

```

CONST

```

```

(* Memory Requirement Types *)
MemPublic = 0; (* Memory will be used by different tasks or interrupt code *)
MemChip   = 1; (* Memory within the range of the special Amiga chips *)
MemFast   = 2; (* Memory outside the range of the special Amiga chips *)
MemClear  = 16; (* Indicates clear memory to zero before returning *)
MemLargest = 17; (* Get largest available memory block *)
MemFailed = 31; (* Returned by func AllocEntry when memory req fails *)

```

```

(* Alignment rules for a memory block *)
MemBlockSize = 8;
MemBlockMask = MemBlockSize-1;

```

```

TYPE

```

```

MemReqSet = SET OF [0..31];

```

```

MemChunkPtr = POINTER TO MemChunk;
MemChunk = RECORD

```

```

    mcNext : MemChunkPtr; (* pointer to next chunk *)
    mcBytes : LONGCARD;    (* chunk byte size *)
END;

```

```

MemHeaderPtr = POINTER TO MemHeader;
MemHeader = RECORD

```

```

    mhNode      : Node;
    mhAttributes : BITSET; (* characteristics of this region *)
    mhFirst     : MemChunkPtr; (* first free region *)
    mhLower     : ADDRESS;    (* lower memory bound *)
    mhUpper     : ADDRESS;    (* upper memory bound+1 *)
    mhFree      : LONGCARD;   (* total number of free bytes *)
END;

```

```

MemEntryPtr = POINTER TO MemEntry;

```

```

MemEntry = RECORD
    CASE : CARDINAL OF
        | 0: meReqs : MemReqSet; (* the AllocMem requirements *)
        | 1: meAddr : ADDRESS;    (* the address of this memory region *)
    END;
    meLength : LONGCARD; (* the length of this memory region *)
END;

```

```

MemListPtr = POINTER TO MemList;

```

```

MemList = RECORD
    mlNode      : Node;
    mlNumEntries : CARDINAL; (* the number of entries in this struct *)
    mlME        : ARRAY [0..0] OF MemEntry; (* the first entry *)
END;

```

Module Memory

```
PROCEDURE AddMemList(size: LONGCARD; attributes: MemReqSet; pri: LONGINT;  
                    base: ADDRESS; name: ADDRESS): LONGCARD;  
(* Add memory to the system free pool.
```

size - the size (in bytes) of the memory area
attributes - the attributes word that the memory pool will have
pri - the priority for this memory. CHIP memory has a pri of -10, extended memory has a priority of 0
base - the base of the new memory area
name - the name that will be used in the memory header, or NIL if no name is provided. This name is not copied so it must remain valid for as long as the memory header is in the system.

result - error *)

```
PROCEDURE AllocAbs(byteSize: LONGCARD; location: ADDRESS): ADDRESS;  
(* Allocate at a given location.
```

byteSize - the size of the desired block in bytes
location - the address where the memory is MUST be

result - a pointer to the allocated free block or NIL if not allocated *)

```
PROCEDURE Allocate(VAR freeList: MemHeader; byteSize: LONGCARD): ADDRESS;  
(* Allocate a block of memory.
```

freeList - the memory list header
byteSize - the size of the desired block in bytes

result - a pointer to the just allocated free block, returns NIL if no free regions large enough to satisfy the request *)

```
PROCEDURE AllocEntry(memList: ADDRESS): ADDRESS;  
(* Allocate many regions of memory.
```

memList - a pointer to a MemList structure filled in with MemEntry structures

result - a different MemList filled in with the actual memory allocated. If enough memory cannot be obtained, then the memory that was allocated is freed. The result is the "meReqs" field of the allocation which failed and the MemFailed bit is also set. Example:
memKey := AllocEntry(memTable);
IF MemFailed IN MemReqSet(memKey) THEN
 (* Abort no memory *)
 IF MemChip IN MemReqSet(memKey) THEN
 WriteString("No CHIP Memory!");
 ELSIF MemFast IN MemReqSet(memKey) THEN
 WriteString("No FAST Memory!");
 END;
 HALT;
END;
(* program... *)
FreeEntry(memKey); (* Release memory *) *)

```
PROCEDURE AllocMem(byteSize: LONGCARD; requirements: MemReqSet): ADDRESS;  
(* Allocate memory given certain requirements.
```

byteSize - the size of desired block in bytes. This number is rounded up to the next larger block size for the actual allocation
requirements - the type of memory block to allocate

result - a pointer to the allocated free block. If block could not be allocated return NIL *)

```
PROCEDURE AvailMem(requirements: MemReqSet): LONGCARD;  
(* Memory available given certain requirements.
```

requirements - a requirements mask as specified in AllocMem

result - total free space remaining *)

```
PROCEDURE CopyMem(source, dest: ADDRESS; size: LONGCARD);
(* General purpose memory copy routine.
```

```
    source - a pointer to the source data region
    dest - a pointer to the destination data region
    size - the size (in bytes) of the memory area *)
```

```
PROCEDURE CopyMemQuick(source, dest: ADDRESS; size: LONGCARD);
(* Optimized memory copy routine. Restrictions: the source and dest pointers
    must be longword aligned, the size must be a multiple of longwords.
```

```
    source - a pointer to the source data region, long aligned
    dest - a pointer to the destination data region, long aligned
    size - the size (in bytes) of the memory area *)
```

```
PROCEDURE Deallocate(VAR freeList: MemHeader; memoryBlock: ADDRESS;
    byteSize: LONGCARD);
(* Deallocate a block of memory.
```

```
    freeList - the free list
    memoryBlock - memory block to return
    byteSize - the size of the desired block in bytes *)
```

```
PROCEDURE FreeEntry(memList: ADDRESS);
(* Free many regions of memory.
```

```
    memList - pointer to a structure filled in with memEntry structures *)
```

```
PROCEDURE FreeMem(memoryBlock: ADDRESS; byteSize: LONGCARD);
(* Deallocate with knowledge.
```

```
    memoryBlock - memory block to free
    byteSize - the size of the block in bytes *)
```

```
PROCEDURE InitStruct(initTable: ADDRESS; memory: ADDRESS; size: CARDINAL);
(* Initialize memory from a table. The format for the initTable is described
    in detail in the autodocs.
```

```
    initTable - the beginning of the commands and data to init memory with. Must
                  be on an even boundary unless only byte initialization is done
    memory - the beginning of the memory to initialize. Must be on an even
              boundary if size is specified
    size - the size of memory, which is used to clear it before initializing
           via initTable. If the size is zero memory is not cleared *)
```

```
PROCEDURE TypeOfMem(address: ADDRESS): MemReqSet;
(* Determine attributes of a given memory address.
```

```
    address - a memory address
```

```
    result - returns with the bits set which describe the memory block, returns
              empty set {}, if the address is not in known RAM *)
```

```
END Memory.
```

Module MiscResource

```
(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: MiscResource.DEF          Version: Amiga.00.00
* Created: 11/29/86   Updated: 11/29/86   Author: Leon Frenkel
* Description: Amiga Misc resource.
*****)
```

DEFINITION MODULE MiscResource;

FROM Libraries IMPORT Library, LibBase, LibVectSize;

CONST

MiscName = "misc.resource";

MRSerialPort = 0;
MRSerialBits = 1;
MRParallelPort = 2;
MRParallelBits = 3;

NumMRTypes = 4;

TYPE

MiscResourcePtr = POINTER TO MiscResource;

MiscResource = RECORD

 mrLibrary : Library;
 mrAllocArray : ARRAY [0..NumMRTypes-1] OF LONGCARD;
END;

CONST

MRAllocMiscResource = LibBase;

MRFreeMiscResource = LibBase + LibVectSize;

END MiscResource.


```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: NarratorDevice.DEF      Version: Amiga.00.00      *
* Created: 11/27/86   Updated: 01/17/87   Author: Leon Frenkel *
* Description: "narrator.device" types.                *
*****)

```

```

DEFINITION MODULE NarratorDevice;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM IODevices IMPORT IOStdReq;

```

```

CONST

```

```

(* Error Codes *)
NDNoMem      = -2; (* can't allocate memory *)
NDNoAudLib   = -3; (* can't open audio device *)
NDMakeBad    = -4; (* error in MakeLibrary call *)
NDUnitErr    = -5; (* unit other than 0 *)
NDCantAlloc  = -6; (* can't allocate audio channel(s) *)
NDUnimpl     = -7; (* unimplemented command *)
NDNoWrite    = -8; (* read for mouth without write first *)
NDExpunged   = -9; (* can't open, deferred expunge bit set *)
NDPhonErr    = -20; (* phoneme code spelling error *)
NDRateErr    = -21; (* rate out of bounds *)
NDPitchErr   = -22; (* pitch out of bounds *)
NDSexErr     = -23; (* sex not valid *)
NDModeErr    = -24; (* mode not valid *)
NDFreqErr    = -25; (* sampling frequency out of bounds *)
NDVolErr     = -26; (* volume out of bounds *)

```

```

(* Input parameters and defaults *)

```

```

DefPitch     = 110;      (* default pitch *)
DefRate      = 150;      (* default speaking rate (wpm) *)
DefVol       = 64;       (* default volume (full) *)
DefFreq      = 22200;    (* default sampling frequency (Hz) *)
Male         = 0;        (* male vocal tract *)
Female       = 1;        (* female vocal tract *)
NaturalFO    = 0;        (* natural pitch contours *)
RoboticFO    = 1;        (* monotone *)
DefSex       = Male;     (* default sex *)
DefMode      = NaturalFO; (* default mode *)

```

```

(* Parameter bounds *)

```

```

MinRate      = 40;      (* minimum speaking rate *)
MaxRate      = 400;     (* maximum speaking rate *)
MinPitch     = 65;      (* minimum pitch *)
MaxPitch     = 320;     (* maximum pitch *)
MinFreq      = 5000;    (* minimum sampling frequency *)
MaxFreq      = 28000;   (* maximum sampling frequency *)
MinVol       = 0;       (* minimum volume *)
MaxVol       = 64;      (* maximum volume *)

```

Module NarratorDevice

TYPE

(* Standard Write request *)

narratorrbPtr = POINTER TO narratorrb;

narratorrb = RECORD

```

    message : IOSTdReq; (* standard IORB *)
    rate    : CARDINAL; (* speaking rate (words/minute) *)
    pitch   : CARDINAL; (* baseline pitch in Hertz *)
    mode    : CARDINAL; (* pitch mode *)
    sex     : CARDINAL; (* sex of voice *)
    chmasks : ADDRESS;  (* pointer to audio alloc maps *)
    nmmasks : CARDINAL; (* number of audio alloc maps *)
    volume  : CARDINAL; (* volume. 0 (off) thru 64 *)
    sampfreq : CARDINAL; (* audio sampling freq *)
    mouths  : BYTE;      (* if non-zero, generate mouths *)
    chanmask : BYTE;     (* which ch mask used (internal) *)
    numchan  : BYTE;     (* num ch masks used (internal) *)
    pad     : BYTE;      (* for alignment *)

```

END;

(* Standard Read request *)

mouthrbPtr = POINTER TO mouthrb;

mouthrb = RECORD

```

    voice : narratorrb; (* Speech IORB *)
    width  : BYTE;       (* Width (returned value) *)
    height : BYTE;       (* Height (returned value) *)
    shape  : BYTE;       (* Internal use, do not modify *)
    pad    : BYTE;       (* For alignment *)

```

END;

END NarratorDevice.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Nodes.DEF                      Version: Amiga.00.00      *
* Created: 11/15/86   Updated: 01/16/87   Author: Leon Frenkel   *
* Description: Exec Node types.                                     *
*****)

```

DEFINITION MODULE Nodes;

FROM SYSTEM IMPORT ADDRESS, BYTE;

TYPE

```

(* normal node *)
NodePtr = POINTER TO Node;
Node = RECORD
  lnSucc : NodePtr; (* ptr to next node in the list *)
  lnPred : NodePtr; (* ptr to previous node in the list *)
  lnType : BYTE;    (* defines the type of the node *)
  lnPri : BYTE;    (* specifies the priority of the node *)
  lnName : ADDRESS; (* points the name of the node or NIL *)
END;

```

(* stripped node, no type checking is possible *)

```

MinNodePtr = POINTER TO MinNode;
MinNode = RECORD
  mlnSucc : MinNodePtr;
  mlnPred : MinNodePtr;
END;

```

CONST

```

(* Node Types *)
NTUnknown = BYTE(0);
NTTask = BYTE(1);
NTInterrupt = BYTE(2);
NTDevice = BYTE(3);
NTMsgPort = BYTE(4);
NTMessage = BYTE(5);
NTFreeMsg = BYTE(6);
NTReplyMsg = BYTE(7);
NTResource = BYTE(8);
NTLibrary = BYTE(9);
NTMemory = BYTE(10);
NTSoftInt = BYTE(11);
NTFont = BYTE(12);
NTProcess = BYTE(13);
NTSemaphore = BYTE(14);
NTSignalSem = BYTE(15);

```

END Nodes.

Module ParallelDevice

```
(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: ParallelDevice.DEF      Version: Amiga.00.00
* Created: 11/27/86   Updated: 05/08/87   Author: Leon Frenkel
* Description: "parallel.device" types.
*****)
```

DEFINITION MODULE ParallelDevice;

FROM IODevices IMPORT CmdNonStd, IOStdReq;

CONST
 ParallelName = "parallel.device";

PDCmdQuery = CmdNonStd + 0;
PDCmdSetParams = CmdNonStd + 1;

TYPE
 IOPArrayPtr = POINTER TO IOPArray;
 IOPArray = RECORD
 PTermArray0 : LONGCARD;
 PTermArray1 : LONGCARD;
 END;

CONST
 (* IOStdRequest.ioFlags := IOFlagsSet{ } *)
 IOParQueued = 6; (* rqst-queued bit *)
 IOParAbort = 5; (* rqst-aborted bit *)
 IOParActive = 4; (* rqst-queued-or-current bit *)

TYPE
 ParFlags = (PFO,
 ParEOFMode, (* EOF mode enabled bit *)
 PF2,
 ParRadBoogie, (* (not yet implemented) *)
 PF4,
 ParShared, (* non-exclusive access bit *)
 PF6,
 PF7);
 ParFlagsSet = SET OF ParFlags;

ParStatus = (IOTPSel, (* printer selected bit *)
 IOTPPaperOut, (* paper out bit *)
 IOTPBusy, (* printer in busy toggle bit *)
 IOPTRWDir, (* read=0, write=1 bit *)
 IOPT4, IOPT5, IOPT6, IOPT7);
 ParStatusSet = SET OF ParStatus;

(*****
(* CAUTION !! IF YOU ACCESS the parallel.device, you MUST (!!!!) use *)
(* an IOExtPar-sized structure or you may overlay innocent memory !! *)
*****)
 IOExtParPtr = POINTER TO IOExtPar;
 IOExtPar = RECORD
 IOPar : IOStdReq;
 ioExtFlags : LONGCARD; (* (not used) flag extension area *)
 ioStatus : ParStatusSet; (* status of parallel port *)
 ioParFlags : ParFlagsSet; (* see ParFlags bit defs above *)
 ioPTermArray : IOPArray; (* termination char array *)
 END;

CONST
 ParErrDevBusy = 1;
 ParErrBufTooBig = 2;
 ParErrInvParam = 3;
 ParErrLineErr = 4;
 ParErrNotOpen = 5;
 ParErrPortReset = 6;
 ParErrInitErr = 7;

END ParallelDevice.

```

(*****
*
*          Modula-2 Software Construction Set
*          (c) 1986 by Leon Frenkel
*
*****
* Name: Ports.DEF          Version: Amiga.00.00
* Created: 11/17/86   Updated: 01/16/87   Author: Leon Frenkel
* Description: Exec Ports Types/Functions.
*****)

DEFINITION MODULE Ports;

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM Nodes IMPORT Node;
FROM Lists IMPORT List;
FROM Tasks IMPORT TaskPtr;

CONST
  (* mpFlags defs *)
  PFAction = BYTE(3); (* sub-field of mpFlags field *)
  PASignal = BYTE(0); (* signal a specified task on the arrival of a new msg *)
  PASoftInt = BYTE(1); (* cause the specified software interrupt *)
  PAIgnore = BYTE(2); (* perform no operation other than queuing the message *)

TYPE
  MsgPortPtr = POINTER TO MsgPort;
  MsgPort = RECORD
    mpNode      : Node;
    mpFlags     : BYTE;
    mpSigBit    : BYTE; (* signal bit number *)
    mpSigTask   : TaskPtr; (* task to be signalled *)
    mpMsgList   : List; (* message linked list *)
  END;

  MessagePtr = POINTER TO Message;
  Message = RECORD
    mnNode      : Node;
    mnReplyPort : MsgPortPtr; (* message reply port *)
    mnLength    : CARDINAL; (* message len in bytes *)
  END;

PROCEDURE AddPort(VAR port: MsgPort);
(* Add a message port to the system.

   port - pointer to a message port *)

PROCEDURE FindPort(name: ADDRESS): MsgPortPtr;
(* Find a given system message port.

   name - name of the port to find

   result - a pointer to the message port, or NIL if not found *)

PROCEDURE GetMsg(VAR port: MsgPort): ADDRESS;
(* Get next message from a message port.

   port - the receiver message port

   result - a pointer to the first message available. If there are no messages
   return NIL *)

PROCEDURE PutMsg(VAR port: MsgPort; message: ADDRESS);
(* Put a message to a message port.

   port - a message port
   message - a message (actual structure varies) *)

```

Module Ports

PROCEDURE RemPort(VAR port: MsgPort);
(* Remove a message port from the system.

port - pointer to a message port *)

PROCEDURE ReplyMsg(message: ADDRESS);
(* Put a message to its reply port.

message - a pointer to the message (actual structure varies) *)

PROCEDURE WaitPort(VAR port: MsgPort): ADDRESS;
(* Wait for a given port to be non-empty.

port - a pointer to the message port

result - a pointer to the first available message *)

END Ports.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: PortsUtil.DEF      Version: Amiga.00.00 *
* Created: 01/04/87   Updated: 01/13/87   Author: Leon Frenkel *
* Description: Exec Support Procedures for Ports. *
*****)

```

```
DEFINITION MODULE PortsUtil;
```

```
FROM SYSTEM IMPORT ADDRESS;
FROM Ports IMPORT MsgPort, MsgPortPtr;
```

```
(* $L+ *)
```

```
PROCEDURE CreatePort(name: ADDRESS; pri: INTEGER): MsgPortPtr;
```

```
(* Allocate memory and a signal for and then initialize a new message port.
```

```
    name - if public port then ptr to a null-terminated string, if private port then
           pass NIL
```

```
    pri - priority to be assigned to the new port
```

```
    result - ptr to a new message port or NIL if unsuccessful creation *)
```

```
PROCEDURE DeletePort(VAR port: MsgPort);
```

```
(* Frees up a message port as allocated by CreatePort().
```

```
    port - port to be deleted *)
```

```
END PortsUtil.
```

Module PotgoResource

```
(*****
*                               Modula-2 Software Construction Set          *
*                               (c) 1986 by Leon Frenkel                    *
*****
* Name: PotgoResource.DEF          Version: Amiga.00.00                  *
* Created: 11/29/86   Updated: 11/29/86   Author: Leon Frenkel          *
* Description: Amiga "potgo.resource" definitions.                      *
*****)
```

```
DEFINITION MODULE PotgoResource;
```

```
FROM Resources IMPORT ResourcePtr;
```

```
CONST
```

```
  PotgoName = "potgo.resource";
```

```
VAR
```

```
  PotgoBase : ResourcePtr;
```

```
PROCEDURE AllocPotBits(bits: BITSET): BITSET;
```

```
(* Allocate bits in the potgo register.
```

```
  bits - a description of the hardware bits the application wishes to allocate
```

```
  result - the bits which were allocated *)
```

```
PROCEDURE FreePotBits(allocated: BITSET);
```

```
(* Free allocated bits in the potgo register.
```

```
  allocated - bits returned by AllocPotBits() *)
```

```
PROCEDURE WritePotgo(word: BITSET; mask: BITSET);
```

```
(* Write to the hardware potgo register.
```

```
  word - the data to write to the hardware potgo register
```

```
  mask - the bits in word that are to be written *)
```

```
END PotgoResource.
```



```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel      *
*                               *
*****
* Name: Preferences.DEF      Version: Amiga.00.00      *
* Created: 11/26/86      Updated: 11/01/87      Author: Leon Frenkel      *
* Description: Amiga Intuition Preferences types/functions.      *
*****)

```

```

DEFINITION MODULE Preferences;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT IntuitionBase; (* Used by IMPLEMENTATION MODULE *)
FROM TimerDevice IMPORT timeval;

```

```

CONST

```

```

(* these are the definitions for the printer configurations *)
FileNameSize = 30; (* Filename size *)

```

```

PointerSize = (1 + 16 + 1) * 2; (* Size of Pointer data buffer *)

```

```

(* These defines are for the default font size. These actually describe the
* height of the default fonts. The default font type is the topaz
* font, which is a fixed width font that can be used in either
* eighty-column or sixty-column mode. The Preferences structure reflects
* which is currently selected by the value found in the variable FontSize,
* which may have either of the values defined below. These values actually
* are used to select the height of the default font. By changing the
* height, the resolution of the font changes as well.
*)

```

```

TopazEighty = 8;

```

```

TopazSixty = 9;

```

```

TYPE

```

```

PreferencesPtr = POINTER TO Preferences;

```

```

Preferences = RECORD

```

```

    (* the default font height *)
    FontHeight      : BYTE; (* height for system default font *)

```

```

    (* constant describing what's hooked up to the port *)
    PrinterPort      : BYTE; (* printer port connection *)

```

```

    (* the baud rate of the port *)
    BaudRate         : CARDINAL; (* baud rate for the serial port *)

```

```

    (* various timing rates *)
    KeyRptSpeed      : timeval; (* repeat speed for keyboard *)
    KeyRptDelay       : timeval; (* Delay before keys repeat *)
    DoubleClick       : timeval; (* Interval allowed between clicks *)

```

```

    (* Intuition Pointer data *)
    PointerMatrix     : ARRAY [0..PointerSize-1] OF CARDINAL;
    XOffset           : BYTE; (* X-Offset for active 'bit' *)
    YOffset           : BYTE; (* Y-Offset for active 'bit' *)
    color17           : CARDINAL; (* Colors for sprite pointer *)
    color18           : CARDINAL;
    color19           : CARDINAL;
    PointerTicks       : CARDINAL; (* Sensitivity of the pointer *)

```

```

    (* Workbench Screen colors *)
    color0            : CARDINAL; (*****
    color1            : CARDINAL; (* Standard default colours *)
    color2            : CARDINAL; (* Used in the Workbench *)
    color3            : CARDINAL; (*****

```

```

    (* positioning data for the Intuition View *)
    ViewXOffset       : BYTE; (* Offset for top lefthand pos *)
    ViewYOffset       : BYTE; (* X and Y dimensions *)
    ViewInitX         : INTEGER; (* View initial x offset *)
    ViewInitY         : INTEGER; (* View initial y offset *)

```

```

    EnableCLI         : INTEGER; (* CLI availability switch *)

```

Module Preferences

```
(* printer configurations *)
PrinterType : CARDINAL; (* printer type *)
PrinterFilename : ARRAY [0..FileNameSize-1] OF CHAR; (* file*)
```

```
(* print format and quality configurations *)
PrintPitch : CARDINAL; (* printer pitch *)
PrintQuality : CARDINAL; (* print quality *)
PrintSpacing : CARDINAL; (* number of lines per inch *)
PrintLeftMargin : CARDINAL; (* left margin in characters *)
PrintRightMargin : CARDINAL; (* right margin in characters *)
PrintImage : CARDINAL; (* positive or negative *)
PrintAspect : CARDINAL; (* horizontal or vertical *)
PrintShade : CARDINAL; (* b&w, half-tone, or color *)
PrintThreshold : INTEGER; (* darkness ctrl for b/w dumps*)
```

```
(* print paper descriptors *)
PaperSize : CARDINAL; (* paper size *)
PaperLength : CARDINAL; (* paper length in lines *)
PaperType : CARDINAL; (* continuous or single sheet *)
```

```
(* Serial device settings: These are six nibble-fields in
* three bytes (these look a little strange so the defaults
* will map out to zero)
*)
```

```
SerRWBits : BYTE; (*upper = 8-number of read bits *)
(*lower = 8-number of write bits *)
SerStopBuf : BYTE; (*upper = number of stop bits-1 *)
(*lower = table value for ButSize *)
SerParShk : BYTE; (*lower = value for Parity setting *)
(*upper = value for handshake mode *)
LaceWB : BYTE; (* if workbench is to be interlaced*)
```

```
(* temp file for printer *)
WorkName : ARRAY [0..FileNameSize-1] OF CHAR;
```

```
(* for further system expansion *)
padding : ARRAY [0..15] OF BYTE;
END;
```

CONST

```
(* Workbench Interlace (use one bit) *)
LaceWB = BYTE(01H);
```

```
(* PrinterPort *)
ParallelPrinter = BYTE(00H);
SerialPrinter = BYTE(01H);
```

```
(* BaudRate *)
Baud110 = 00H;
Baud300 = 01H;
Baud1200 = 02H;
Baud2400 = 03H;
Baud4800 = 04H;
Baud9600 = 05H;
Baud19200 = 06H;
BaudMIDI = 07H;
```

```
(* PaperType *)
FanFold = 00H;
Single = 80H;
```

```
(* PrintPitch *)
Pica = 000H;
Elite = 400H;
Fine = 800H;
```

```
(* PrintQuality *)
Draft = 000H;
Letter = 100H;
```

```
(* PrintSpacing *)
SixLPI = 000H;
EightLPI = 200H;
```

```
(* PrintImage *)
ImagePositive = 00H;
ImageNegative = 01H;
```

```
(* PrintAspect *)
AspectHoriz = 00H;
AspectVert = 01H;
```

```
(* PrintShade *)
ShadeBW = 00H;
ShadeGreyScale = 01H;
ShadeColor = 02H;
```

```
(* PaperSize *)
USLetter = 00H;
USLegal = 10H;
NTractor = 20H;
WTractor = 30H;
Custom = 40H;
```

```
(* PrinterType *)
CustomName = 00H;
AlphaP101 = 01H;
Brother15XL = 02H;
CBMMPS1000 = 03H;
Diab630 = 04H;
DiabAdvD25 = 05H;
DiabC150 = 06H;
Epson = 07H;
EpsonJX80 = 08H;
Okimate20 = 09H;
QumeLP20 = 0AH;
HPLaserJet = 0BH;
HPLaserJetPlus = 0CH;
```

```
(* Serial Input Buffer Sizes *)
SBuf512 = BYTE(00H);
SBuf1024 = BYTE(01H);
SBuf2048 = BYTE(02H);
SBuf4096 = BYTE(03H);
SBuf8000 = BYTE(04H);
SBuf16000 = BYTE(05H);
```

```
(* Serial Bit Masks *)
SReadBits = BYTE(0FOH); (* for SerRWBits *)
SWriteBits = BYTE(00FH);
```

```
SStopBits = BYTE(0FOH); (* for SerStopBuf *)
SBufSizeBits = BYTE(00FH);
```

```
SParityBits = BYTE(0FOH); (* for SerParShk *)
SShakeBits = BYTE(00FH);
```

```
(* Serial Parity in upper nibble *)
SParityNone = BYTE(0);
SParityEven = BYTE(1);
SParityOdd = BYTE(2);
```

```
(* Serial HandShake Mode in lower nibble *)
SHShakeXON = BYTE(0);
SHShakeRTS = BYTE(1);
SHShakeNone = BYTE(2);
```

```
PROCEDURE GetDefPrefs(PrefBuffer: ADDRESS; Size: INTEGER): ADDRESS;
(* Get a copy of the Intuition default Preferences.
```

PrefBuffer - pointer to the memory buffer to receive your copy of preferences
 Size - the number of bytes in your PrefBuffer, the number of bytes you want
 copied from the system's internal Preference settings

result - returns your parameter PrefBuffer *)

Module Preferences

PROCEDURE GetPrefs(PrefBuffer: ADDRESS; Size: INTEGER): ADDRESS;
(* Get the current setting of the Intuition Preferences.

PrefBuffer - pointer to the memory buffer to receive your copy of preferences
Size - the number of bytes in your PrefBuffer, the number of bytes you want
copied from the system's internal Preference settings

result - returns your parameter PrefBuffer *)

PROCEDURE SetPrefs(PrefBuffer: ADDRESS; Size: INTEGER; Inform: BOOLEAN): ADDRESS;
(* Set new preference values.

PrefBuffer - pointer to the memory buffer which contains your desired setting
for Intuition Preferences
Size - the number of bytes in your PrefBuffer, the number of bytes you want
copied to the system's internal Preference settings
Inform - whether you want the information of a new Preferences setting
propagated to all windows

result - returns your parameter PrefBuffer *)

END Preferences.

```

*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: PrinterBase.DEF                      Version: Amiga.00.00      *
* Created: 08/26/87   Updated: 08/26/87   Author: Leon Frenkel        *
* Description: Printer device data definition.                          *
*****

```

```
DEFINITION MODULE PrinterBase;
```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM AmigaDOS IMPORT BPTR;
FROM Libraries IMPORT Library;
FROM ParallelDevice IMPORT IOExtPar;
FROM Ports IMPORT MsgPort;
FROM Preferences IMPORT Preferences;
FROM SerialDevice IMPORT IOExtSer;
FROM Tasks IMPORT Task;
FROM TimerDevice IMPORT timerequest;

```

```
TYPE
```

```

DeviceDataPtr = POINTER TO DeviceData;
DeviceData = RECORD
    ddDevice      : Library; (* standard library node *)
    ddSegment     : ADDRESS;  (* A0 when initialized *)
    ddExecBase    : ADDRESS;  (* A6 for exec *)
    ddCmdVectors  : ADDRESS;  (* command table for device cmds *)
    ddCmdBytes    : ADDRESS;  (* bytes desc which cmds queue *)
    ddNumCommands : CARDINAL; (* number of cmds supported *)
END;

```

```
CONST
```

```

PstkSize = 800H;

PPCGfx  = BYTE(0);
PPCColor = BYTE(1);

PPCBWAlpha = BYTE(0); (* black&white alphanumerics *)
PPCBWGfx   = BYTE(1); (* black&white graphics *)
PPCColorGfx = BYTE(3); (* color graphics *)

PPCBW    = BYTE(1); (* only black&white *)
PPCYMC   = BYTE(2); (* only yellow/magenta/cyan *)
PPCYMCBW = BYTE(3); (* yellow/magenta/cyan or black&white *)
PPCYMCB  = BYTE(4); (* yellow/magenta/cyan/black *)

PPC4Color = BYTE(4); (* a flag for YMCB and BGRW *)
PPCAdditive = BYTE(8); (* not yellow/magenta/cyan/black, *)
               (* but blue/green/red/white *)
PPCWB     = BYTE(9); (* only black&white, 0 == BLACK *)
PPCBGR    = BYTE(0AH); (* blue/green/red *)
PPCBGRWB  = BYTE(0BH); (* blue/green/red or black&white *)
PPCBGRW   = BYTE(0CH); (* blue/green/red/white *)

```

Module PrinterBase

TYPE

PrinterExtendedDataPtr = POINTER TO PrinterExtendedData;

PrinterExtendedData =

RECORD

```

pedPrinterName : ADDRESS; (* printer name, null terminated *)
pedInit        : ADDRESS; (* called after LoadSeg *)
pedExpunge     : ADDRESS; (* called before LoadSeg *)
pedOpen        : ADDRESS; (* called at OpenDevice *)
pedClose       : ADDRESS; (* called at CloseDevice *)
pedPrinterClass : BYTE;    (* printer class *)
pedColorClass  : BYTE;    (* color class *)
pedMaxColumns  : BYTE;    (* number of print columns available *)
pedNumCharSets : BYTE;    (* number of character sets *)
pedNumRows     : CARDINAL; (* number of raster rows in a raster dump *)
pedMaxXDots    : LONGCARD; (* number of dots maximum in a raster dump *)
pedMaxYDots    : LONGCARD; (* number of dots maximum in a raster dump *)
pedXDotsInch   : CARDINAL; (* horizontal dot density *)
pedYDotsInch   : CARDINAL; (* vertical dot density *)
pedCommands    : ADDRESS; (* printer text command table *)
pedDoSpecial   : ADDRESS; (* special command handler *)
pedRender      : ADDRESS; (* raster render function *)
pedTimeoutSecs : LONGINT;  (* good write timeout *)
(* the following only exists if the segment version is 33 or greater *)
ped8BitChars   : ADDRESS; (* conversion strings for the extended font *)

```

END;

PrinterSegmentPtr = POINTER TO PrinterSegment;

PrinterSegment = RECORD

```

psNextSegment : LONGCARD; (* (actually a BPTR) *)
psrunAlert    : LONGCARD; (* MOVEQ #0,DO : RTS *)
psVersion     : CARDINAL; (* segment version *)
psRevision    : CARDINAL; (* segment revision *)
psPED         : PrinterExtendedData; (* printe ext data *)

```

END;

PrinterDataPtr = POINTER TO PrinterData;

PrinterData = RECORD

```

pdDevice      : DeviceData;
pdUnit        : MsgPort; (* the one and only unit *)
pdPrinterSegm : BPTR;    (* the printer specific segment *)
pdPrinterType : CARDINAL; (* the segment printer type *)
pdSegmentData : PrinterSegmentPtr; (* segment data struc *)
pdPrintBuf    : ADDRESS; (* the raster print buffer *)
pdPWrite      : ADDRESS; (* the write function *)
pdPBothReady  : ADDRESS; (* write function's done *)
CASE : CARDINAL OF (* port I/O request 0 *)
| 0: pdPIORO : IOExtPar;
| 1: pdPSIORO : IOExtSer;
END;
CASE : CARDINAL OF (* and 1 for double buffering *)
| 0: pdPIOR1 : IOExtPar;
| 1: pdPSIOR1 : IOExtSer;
END;
pdTIOR        : timerequest; (* timer I/O request *)
pdIORPort     : MsgPort;    (* and message reply port *)
pdTC          : Task;       (* write task *)
pdStk         : ARRAY [0..PStkSize-1] OF BYTE; (* stack space *)
pdFlags       : BYTE;       (* device flags *)
pdpad         : BYTE;
pdPreferences : Preferences; (* the latest preferences *)
pdPWaitEnabled : BYTE;      (* wait function switch *)

```

END;

END PrinterBase.


```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: PrinterDevice.DEF      Version: Amiga.00.00
* Created: 11/27/86      Updated: 11/27/86      Author: Leon Frenkel
* Description: "printer.device" types.
*****)

```

```
DEFINITION MODULE PrinterDevice;
```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM IODevices IMPORT CmdNonStd, IOFlagsSet, IORequest, DevicePtr, UnitPtr;
FROM Ports IMPORT Message;
FROM Rasters IMPORT RastPortPtr;
FROM Views IMPORT ColorMapPtr;

```

```
CONST
```

```

PRDRawWrite   = CmdNonStd + 0;
PRDPrtCommand = CmdNonStd + 1;
RPDDumpRPort  = CmdNonStd + 2;

```

```
(* printer command definitions *)
```

```

aRIS  = 0; (* ESCc reset                ISO *)
aRIN  = 1; (* ESC#1 initialize            +++ *)
aIND  = 2; (* ESCD lf                    ISO *)
aNEL  = 3; (* ESCe return,lf             ISO *)
aRI   = 4; (* ESCM reverse lf            ISO *)

```

```

aSGRO = 5; (* ESC[0m normal char set      ISO *)
aSGR3 = 6; (* ESC[3m italics on           ISO *)
aSGR23 = 7; (* ESC[23m italics off        ISO *)
aSGR4 = 8; (* ESC[4m underline on         ISO *)
aSGR24 = 9; (* ESC[24m underline off      ISO *)
aSGR1 = 10; (* ESC[1m boldface on         ISO *)
aSGR22 = 11; (* ESC[22m boldface off      ISO *)
aSFC = 12; (* SGR30-39 set foreground color ISO *)
aSBC = 13; (* SGR40-49 set background color ISO *)

```

```

aSHORP0 = 14; (* ESC[0w normal pitch      DEC *)
aSHORP2 = 15; (* ESC[2w elite on           DEC *)
aSHORP1 = 16; (* ESC[1w elite off         DEC *)
aSHORP4 = 17; (* ESC[4w condensed fine on  DEC *)
aSHORP3 = 18; (* ESC[3w condensed off     DEC *)
aSHORP6 = 19; (* ESC[6w enlarged on      DEC *)
aSHORP5 = 20; (* ESC[5w enlarged off     DEC *)

```

```

aDEN6 = 21; (* ESC[6*z shadow print on    DEC (sort of) *)
aDEN5 = 22; (* ESC[5*z shadow print off   DEC *)
aDEN4 = 23; (* ESC[4*z doublestrike on    DEC *)
aDEN3 = 24; (* ESC[3*z doublestrike off   DEC *)
aDEN2 = 25; (* ESC[2*z NLQ on             DEC *)
aDEN1 = 26; (* ESC[1*z NLQ off           DEC *)

```

```

aSUS2 = 27; (* ESC[2v superscript on      +++ *)
aSUS1 = 28; (* ESC[1v superscript off     +++ *)
aSUS4 = 29; (* ESC[4v subscript on        +++ *)
aSUS3 = 30; (* ESC[3v subscript off      +++ *)
aSUS0 = 31; (* ESC[0v normalize the line  +++ *)
aPLU = 32; (* ESCL partial line up       ISO *)
aPLD = 33; (* ESCK partial line down     ISO *)

```

```

aFNT0 = 34; (* ESC(B US char set          DEC *)
aFNT1 = 35; (* ESC(R French char set      DEC *)
aFNT2 = 36; (* ESC(K German char set     DEC *)
aFNT3 = 37; (* ESC(A UK char set         DEC *)
aFNT4 = 38; (* ESC(E Danish I char set   DEC *)
aFNT5 = 39; (* ESC(H Sweden char set     DEC *)
aFNT6 = 40; (* ESC(Y Italian char set    DEC *)
aFNT7 = 41; (* ESC(Z Spanish char set    DEC *)
aFNT8 = 42; (* ESC(J Japanese char set   +++ *)
aFNT9 = 43; (* ESC(6 Norwegian char set  DEC *)
aFNT10 = 44; (* ESC(C Danish II char set  +++ *)

```

Module PrinterDevice

```

aPROP2 = 45; (* ESC[2p proportional on      +++ *)
aPROP1 = 46; (* ESC[1p proportional off     +++ *)
aPROPO = 47; (* ESC[Op proportional clear   +++ *)
aTSS = 48; (* ESC[n E set proportional offset ISO *)
aJFY5 = 49; (* ESC[5 F auto left justify   ISO *)
aJFY7 = 50; (* ESC[7 F auto right justify  ISO *)
aJFY6 = 51; (* ESC[6 F auto full justify   ISO *)
aJFY0 = 52; (* ESC[0 F auto justify off    ISO *)
aJFY3 = 53; (* ESC[3 F letter space (justify) ISO (special) *)
aJFY1 = 54; (* ESC[1 F word fill(auto center) ISO (special) *)

```

```

aVERPO = 55; (* ESC[Oz 1/8" line spacing   +++ *)
aVERP1 = 56; (* ESC[1z 1/6" line spacing   +++ *)
aSLPP = 57; (* ESC[nt set form length n     DEC *)
aPERF = 58; (* ESC[nq perf skip n (n>0)    +++ *)
aPERFO = 59; (* ESC[Oq perf skip off       +++ *)

```

```

aLMS = 60; (* ESC#9 Left margin set        +++ *)
aRMS = 61; (* ESC#0 Right margin set        +++ *)
aTMS = 62; (* ESC#8 Top margin set          +++ *)
aBMS = 63; (* ESC#2 Bottom marg set         +++ *)
aSTBM = 64; (* ESC[Pn1;Pn2r T&B margins     DEC *)
aSLRM = 65; (* ESC[Pn1;Pn2s L&R margin      DEC *)
aCAM = 66; (* ESC#3 Clear margins           +++ *)

```

```

aHTS = 67; (* ESC# Set horiz tab           ISO *)
aVTS = 68; (* ESC# Set vertical tabs       ISO *)
aTBCO = 69; (* ESC[Og Clr horiz tab        ISO *)
aTBC3 = 70; (* ESC[3g Clear all h tab      ISO *)
aTBC1 = 71; (* ESC[1g Clr vertical tabs    ISO *)
aTBC4 = 72; (* ESC[4g Clr all v tabs       ISO *)
aTBCALL = 73; (* ESC#4 Clr all h & v tabs  +++ *)
aTBSALL = 74; (* ESC#5 Set default tabs    +++ *)
aEXTEND = 75; (* ESC[Pn*x extended commands +++ *)

```

TYPE

```

IOPrtCmdReqPtr = POINTER TO IOPrtCmdReq;
IOPrtCmdReq = RECORD

```

```

    ioMessage      : Message;
    ioDevice       : DevicePtr; (* device not pointer *)
    ioUnit         : UnitPtr;   (* unit (driver private) *)
    ioCommand      : CARDINAL; (* device command *)
    ioFlags        : IOFlagsSet;
    ioError        : BYTE;      (* error or warning num *)
    ioPrtCommand   : CARDINAL; (* printer command *)
    ioParam0       : BYTE;      (* first command parameter *)
    ioParam1       : BYTE;      (* second command parameter *)
    ioParam2       : BYTE;      (* third command parameter *)
    ioParam3       : BYTE;      (* fourth command parameter *)
END;

```

```

Special = (SpecialMtlCols, (* DestCols specified in 1/1000 *)
            SpecialMtlRows, (* DestRows specified in 1/1000 *)
            SpecialFullCols, (* make DestCols maximum possible *)
            SpecialFullRows, (* make DestRows maximum possible *)
            SpecialFracCols, (* DestCols is fraction of FullCols *)
            SpecialFracRows, (* DestRows is fraction of FullRows *)
            SpecialCenter,   (* center image *)
            SpecialAspect,   (* ensure correct aspect ratio *)
            SF8, SF9, SF10, SF11, (* reserved *)
            SF12, SF13, SF14, SF15); (* Density bits (described below) *)
SpecialSet = SET OF Special;

```

CONST

```

SpecialDensityMask = SpecialSet{SF12..SF15}; (* masks out density bits *)
SpecialDensity1 = SpecialSet{SF12}; (* lowest res *)
SpecialDensity2 = SpecialSet{SF13}; (* next res *)
SpecialDensity3 = SpecialSet{SF12, SF13}; (* next res *)
SpecialDensity4 = SpecialSet{SF14}; (* highest res *)

```


TYPE

IODRPreReqPtr = POINTER TO IODRPreReq;
 IODRPreReq = RECORD

ioMessage : Message;
 ioDevice : DevicePtr; (* device node pointer *)
 ioUnit : UnitPtr; (* unit (driver private) *)
 ioCommand : CARDINAL; (* device command *)
 ioFlags : IOFlagsSet;
 ioError : BYTE; (* error or warning num *)
 ioRastPort : RastPortPtr; (* raster port *)
 ioColorMap : ColorMapPtr; (* color map *)
 ioModes : LONGCARD; (* graphics viewport modes *)
 ioSrcX : CARDINAL; (* source x origin *)
 ioSrcY : CARDINAL; (* source y origin *)
 ioSrcWidth : CARDINAL; (* source x width *)
 ioSrcHeight : CARDINAL; (* source y height *)
 ioDestCols : LONGINT; (* destination x width *)
 ioDestRows : LONGINT; (* destination y height *)
 ioSpecial : SpecialSet; (* option flags *)

END;

CONST

PDErrCancel = 1; (* user canceled a printer timeout *)
 PDErrNotGraphics = 2; (* printer cannot output graphics *)
 PDErrInvertHam = 3; (* cannot invert hold & modify print *)
 PDErrBadDimension = 4; (* print dimensions illegal *)
 PDErrDimensionOvflow = 5; (* print dimensions too large *)
 PDErrInternalMemory = 6; (* no memory for internal variables *)
 PDErrBufferMemory = 7; (* no memory for print buffer *)

END PrinterDevice.

Module RandomNumbers

```
(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: RandomNumbers.DEF      Version: Amiga.00.00
* Created: 01/13/87   Updated: 01/13/87   Author: Leon Frenkel
* Description: Pseudo-Random Number Generator.
*****)
```

```
DEFINITION MODULE RandomNumbers;
```

```
(* $L+*)
```

```
VAR
```

```
Seed: LONGCARD; (* The seed for the random number generator *)
```

```
PROCEDURE Random(max: LONGCARD): LONGCARD;
```

```
(* Return a pseudo-random number between 0 and max-1.
```

max - specifies the upper bounds of the random number range

result - a random number in the specified range *)

```
END RandomNumbers.
```

```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*                               *****
* Name: Rasters.DEF              Version: Amiga.00.00
* Created: 11/20/86   Updated: 05/07/87   Author: Leon Frenkel
* Description: Graphics Raster types/functions.
*****)

```

DEFINITION MODULE Rasters;

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Graphics IMPORT BitMapPtr;

```

TYPE

DrawModeSet = SET OF [0..7];

CONST

```

(* Drawing Modes *)
Jam1      = DrawModeSet{ }; (* jam 1 color into raster *)
Jam2      = DrawModeSet{0}; (* jam 2 colors into raster *)
Complement = DrawModeSet{1}; (* XOR bits into raster *)
InversVid = DrawModeSet{2}; (* inverse video for drawing modes *)

```

TYPE

```

RasInfoPtr = POINTER TO RasInfo;
RasInfo = RECORD
  Next      : RasInfoPtr; (* used for dualpf *)
  BitMap    : BitMapPtr;
  RxOffset  : INTEGER;    (* scroll offsets in this BitMap *)
  RyOffset  : INTEGER;
END;

```

TmpRasPtr = POINTER TO TmpRas;

```

TmpRas = RECORD
  RasPtr : ADDRESS;
  Size   : LONGINT;
END;

```

(* these are the flag bits for RastPort flags *)

```

RastPortFlags = (FrstDot, (* draw the first dot of this line ? *)
  OneDot,      (* use one dot mode for drawing lines *)
  DBuffer,     (* flag set when RastPorts are double-buffered *)
  AreaOutline, (* used by areafiller *)
  RPF4,
  NoCrossFill, (* areafills have no crossovers *)
  RPF6, RPF7,
  RPF8, RPF9,
  RPF10, RPF11,
  RPF12, RPF13,
  RPF14, RPF15);

```

RastPortFlagsSet = SET OF RastPortFlags;

RastPortPtr = POINTER TO RastPort;

```

RastPort = RECORD
  Layer      : ADDRESS; (* LayerPtr, circular reference *)
  BitMap     : BitMapPtr;
  AreaPtrn   : ADDRESS; (* ptr to areafill pattern *)
  TmpRas     : TmpRasPtr;
  AreaInfo   : ADDRESS; (* AreaInfoPtr, circular reference *)
  GelsInfo   : ADDRESS; (* GelsInfoPtr, circular reference *)
  Mask       : BYTE;    (* write mask for this raster *)
  FgPen      : BYTE;    (* foreground pen for this raster *)
  BgPen      : BYTE;    (* background pen *)
  AOPen      : BYTE;    (* areafill outline pen *)
  DrawMode   : DrawModeSet; (* drawing mode for fill, lines, text *)
  AreaPtSz   : BYTE;    (* 2^n words for areafill pattern *)
  Linpatcnt  : BYTE;    (* current line drawing pattern preshift *)
  dummy     : BYTE;
  Flags      : RastPortFlagsSet; (* miscellaneous control bits *)
  LinePtrn   : BITSET;  (* 16 bits for textured lines *)
  cpx        : INTEGER; (* current pen x position *)
  cpy        : INTEGER; (* current pen y position *)
  minterm    : ARRAY [0..7] OF BYTE;

```

Module Rasters

```

PenWidth      : INTEGER;
PenHeight     : INTEGER;
Font          : ADDRESS; (* current font TextFontPtr, circular reference *)
AlgoStyle     : BYTE;    (* the algorithmically generated style *)
TxFlags       : BYTE;    (* text specific flags *)
TxHeight      : CARDINAL; (* text height *)
TxWidth       : CARDINAL; (* text nominal width *)
TxBaseline    : CARDINAL; (* text baseline *)
TxSpacing     : INTEGER; (* text spacing (per character) *)
RPUser        : ADDRESS;
longreserved  : ARRAY [0..1] OF LONGCARD;
wordreserved  : ARRAY [0..6] OF CARDINAL; (* used to be a node *)
reserved      : ARRAY [0..7] OF BYTE;    (* for future use *)
END;
```

```

(* NOTE: *)
(* there is only one style of clipping: raster clipping *)
(* this preserves the continuity of jaggies regardless of clip window *)
(* When drawing into a RastPort, if the ptr to ClipRect is nil then there *)
(* is no clipping done, this is dangerous but useful for speed *)
```

```

PROCEDURE AllocRaster(width, height: CARDINAL): ADDRESS;
(* Allocate space for a bitplane.
```

```

width - number of bits wide for bitplane
height - number of rows in bitplane
```

```

result - pointer to first word in bitplane, if unable to allocate space
        then NIL result will be returned *)
```

```

PROCEDURE FreeRaster(p: ADDRESS; width, height: CARDINAL);
(* Release an allocated area to the system free memory pool.
```

```

p - a pointer to a memory space returned as a result of a call to AllocRaster
width - the width in bits of the bitplane
height - number of rows in bitplane *)
```

```

PROCEDURE InitRastPort(VAR rp: RastPort);
(* Initialize raster port structure.
```

```

rp - a RastPort structure *)
```

```

PROCEDURE InitTmpRas(VAR tmpRas: TmpRas; buffer: ADDRESS; size: LONGINT);
(* Initialize area of local memory for usage by areafill, floodfill, text.
```

```

tmpRas - a TmpRas structure to be linked into a RastPort
buffer - pointer to a contiguous piece of chip memory
size - size in bytes of buffer *)
```

```

PROCEDURE ScrollRaster(VAR rp: RastPort; dx, dy: INTEGER;
                      xmin, ymin: INTEGER; xmax, ymax: INTEGER);
(* Push bits in rectangle in raster around by dx, dy towards 0,0 inside rect.
```

```

rp - a RastPort structure
dx - integer that may be positive, zero, or negative
dy - integer that may be positive, zero, or negative
xmin - x of upper left of bounding rectangle
ymin - y of upper left of bounding rectangle
xmax - x of lower right of bounding rectangle
ymax - y of lower right of bounding rectangle *)
```

```

PROCEDURE SetRast(VAR rp: RastPort; pen: CARDINAL);
(* Set an entire drawing area to a specified color.
```

```

rp - a RastPort structure
pen - the pen number (0..255) to jam into bitmap *)
```

END Rasters.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: RealConversions.DEF      Version: Amiga.00.00      *
* Created: 12/25/86   Updated: 01/19/87   Author: Leon Frenkel *
* Description: Conversion of REAL numbers to strings and conversion of *
* strings to REAL numbers.      *
*****)

```

```

DEFINITION MODULE RealConversions;

```

```

(*$L-*)

```

```

TYPE

```

```

  RealToStringFormat = (Scientific,    (* Scientific - [-]m.nnnnnnnE[+|-]xx *)
                        Decimal,       (* Decimal    - [-]mmm.nnnnnn  *)
                        ShortestForm); (* Use which ever form is shorter *)

```

```

PROCEDURE ConvStringToReal(s: ARRAY OF CHAR): REAL;

```

```

(* Convert a string representation of a floating point number to its REAL val.

```

```

   s - string containing the ascii floating point number

```

```

   result - the floating point number *)

```

```

PROCEDURE ConvRealToString(x: REAL; VAR s: ARRAY OF CHAR;

```

```

   digits: CARDINAL; format: RealToStringFormat);

```

```

(* Convert a REAL number to a string representation.

```

```

   x - number to be converted

```

```

   s - string which is to receive the result

```

```

   digits - number of digits to generate past the decimal point

```

```

   format - the desired floating point format *)

```

```

END RealConversions.

```

Module RealInOut

```
(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: RealInOut.DEF          Version: Amiga.00.00
* Created: 01/14/87   Updated: 07/24/87   Author: Leon Frenkel
* Description: Standard RealInOut module as described in appendix 2 of
* Programming in Modula-2 by N. Wirth.
*****)
```

DEFINITION MODULE RealInOut;

(* \$L+*)

VAR Done: BOOLEAN;

PROCEDURE ReadReal(VAR x: REAL);

(* Read REAL number. Input terminates with a blank or any control character.

x - variable into which to store real number *)

PROCEDURE WriteReal(x: REAL; n: CARDINAL);

(* Write x using n characters. If fewer than n characters are needed, leading blanks are inserted.

x - real value to be output

n - chars to output *)

END RealInOut.

Module Regions

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Regions.DEF                      Version: Amiga.00.00      *
* Created: 11/20/86   Updated: 11/20/86   Author: Leon Frenkel    *
* Description: Graphics Regions types/functions.                  *
*****)

DEFINITION MODULE Regions;

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Graphics IMPORT Rectangle;

TYPE
  RegionRectanglePtr = POINTER TO RegionRectangle;
  RegionRectangle = RECORD
    Next      : RegionRectanglePtr;
    Prev      : RegionRectanglePtr;
    bounds    : Rectangle;
  END;

  RegionPtr = POINTER TO Region;
  Region = RECORD
    bounds      : Rectangle;
    RegionRectangle : RegionRectanglePtr;
  END;

PROCEDURE AndRectRegion(VAR region: Region; VAR rectangle: Rectangle);
(* Perform 2d AND operation of rectangle with region, leaving result in region.

   region - Region structure
   rectangle - Rectangle structure *)

PROCEDURE AndRegionRegion(VAR region1, region2: Region): BOOLEAN;
(* Perform 2d AND operation of one region with second region, leaving result
   in second region.

   region1 - Region structure
   region2 - Region structure to use and for result

   result - TRUE if successful operation, FALSE if ran out of memory *)

PROCEDURE ClearRectRegion(VAR region: Region; VAR rectangle: Rectangle): BOOLEAN;
(* Perform 2d CLEAR operation of rect with region, leaving result in region.

   region - Region structure
   rectangle - Rectangle structure

   result - TRUE if successful operation, FALSE if ran out of memory *)

PROCEDURE ClearRegion(VAR region: Region);
(* Remove all rectangles from region.

   region - Region structure *)

PROCEDURE DisposeRegion(VAR region: Region);
(* Return all space for this region to free memory pool.

   region - Region structure *)

```

Module Regions

```
PROCEDURE NewRegion(): RegionPtr;  
(* Get a clear region.
```

```
    result - pointer to initialized region. if it could not allocate required  
            memory returns NIL *)
```

```
PROCEDURE OrRectRegion(VAR region: Region; VAR rectangle: Rectangle): BOOLEAN;  
(* Perform 2d OR operation of rectangle with region, leaving result in region.
```

```
    region - Region structure  
    rectangle - Rectangle structure
```

```
    result - TRUE if successful operation, FALSE if ran out of memory *)
```

```
PROCEDURE OrRegionRegion(VAR region1, region2: Region): BOOLEAN;  
(* Perform 2d OR operation of one region with second region, leaving  
    result in second region.
```

```
    region1 - Region structure  
    region2 - Region structure
```

```
    result - return TRUE if successful operation, FALSE if ran out of memory *)
```

```
PROCEDURE XorRectRegion(VAR region: Region; VAR rectangle: Rectangle): BOOLEAN;  
(* Perform 2d XOR operation of rectangle with region, leaving result in region.
```

```
    region - Region structure  
    rectangle - Rectangle structure
```

```
    result - return TRUE if successful operation, FALSE if ran out of memory *)
```

```
PROCEDURE XorRegionRegion(VAR region1, region2: Region): BOOLEAN;  
(* Perform 2d XOR operation of one region with second region, leaving  
    result in second region.
```

```
    region1 - Region structure  
    region2 - Region structure
```

```
    result - return TRUE if successful operation, FALSE if ran out of memory *)
```

```
END Regions.
```



```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*****
* Name: Resident.DEF           Version: Amiga.00.00
* Created: 11/19/86   Updated: 07/24/87   Author: Leon Frenkel
* Description: Exec Resident Types/Functions.
*****)

```

```

DEFINITION MODULE Resident;

```

```

FROM SYSTEM IMPORT BYTE, ADDRESS;

```

```

TYPE

```

```

  RTFlagsSet = SET OF [0..7];

```

```

  ResidentPtr = POINTER TO Resident;

```

```

  Resident = RECORD

```

```

    rtMatchWord : CARDINAL; (* word to match on (ILLEGAL) *)
    rtMatchTag  : ResidentPtr; (* pointer to the above *)
    rtEndSkip   : ADDRESS; (* address to continue scan *)
    rtFlags     : RTFlagsSet; (* various tag flags *)
    rtVersion   : BYTE; (* release version number *)
    rtType      : BYTE; (* type of mode (NTmumble) *)
    rtPri       : BYTE; (* initialization priority *)
    rtName      : ADDRESS; (* pointer to node name *)
    rtIdString  : ADDRESS; (* pointer to ident string *)
    rtInit      : ADDRESS; (* pointer to init code *)

```

```

  END;

```

```

CONST

```

```

  RTCMatchWord = 4AFCH;

```

```

  (* flags definitions of Resident.rtFlags field *)

```

```

  RTAutoInit = 7;

```

```

  RTColdStart = 0;

```

```

PROCEDURE FindResident(name: ADDRESS): ResidentPtr;

```

```

  (* Find a resident module by name.

```

```

    name - pointer to a name string

```

```

    result - pointer to the resident tag structure or NIL if none found *)

```

```

PROCEDURE InitCode(startClass, version: LONGCARD);

```

```

  (* Initialize resident code modules.

```

```

    startClass - the class of code to be initialized: coldstart, coolstart,
                  warmstart, ...

```

```

    version - a major version number *)

```

```

PROCEDURE InitResident(VAR resident: Resident; segList: ADDRESS);

```

```

  (* Initialize a resident module.

```

```

    resident - a resident node

```

```

    segList - a list of segments *)

```

```

END Resident.

```

Module Resources

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Resources.DEF      Version: Amiga.00.00
* Created: 11/19/86   Updated: 11/22/86   Author: Leon Frenkel
* Description: Exec Resources Types/Functions.
*****)
```

DEFINITION MODULE Resources;

FROM SYSTEM IMPORT ADDRESS;
FROM Libraries IMPORT Library;

TYPE

ResourcePtr = POINTER TO Resource;
Resource = RECORD
 rLibrary : Library;
END;

PROCEDURE AddResource(VAR resource: Resource);
(* Add a resource to the system.

resource - a properly initialized resource node *)

PROCEDURE OpenResource(resName: ADDRESS): ResourcePtr;
(* Gain access to a resource.

resName - the name of the resource requested

result - if successful, a resource pointer, else NIL *)

PROCEDURE RemResource(VAR resource : Resource);
(* Remove a resource from the system.

resource - a resource node *)

END Resources.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: RomLayers.DEF          Version: Amiga.00.00 *
* Created: 11/24/86   Updated: 11/24/86   Author: Leon Frenkel *
* Description: Graphics Rom Layers functions. *
*****)

DEFINITION MODULE RomLayers;

FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Clipping IMPORT Layer;

PROCEDURE AttemptLockLayerRom(VAR layer: Layer): BOOLEAN;
(* Attempt to Lock Layer structure by rom (gfx lib) code.

   layer - Layer structure

   result - TRUE or FALSE depending on whether the layer is now lock by caller*)

PROCEDURE CopySBitMap(VAR layer: Layer);
(* Synchronize Layer window with contents of Super BitMap.

   layer - SuperBitMap Layer. The Layer must already be locked by the caller *)

PROCEDURE LockLayerRom(VAR layer: Layer);
(* Lock layer structure by rom (gfx lib) code.

   layer - Layer structure *)

PROCEDURE SyncSBitMap(VAR layer: Layer);
(* Synchronize Super BitMap with whatever is in the standard Layer bounds.

   layer - Layer that has a SuperBitMap. The Layer should be locked by caller *)

PROCEDURE UnlockLayerRom(VAR layer: Layer);
(* Unlock Layer structure by rom(gfx lib) code.

   layer - Layer structure *)

END RomLayers.

```

Module RunTimeErrors

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: RunTimeErrors.DEF      Version: Amiga.00.00
* Created: 02/26/87   Updated: 02/26/87   Author: Leon Frenkel
* Description: This module is usefull for debugging purposes. Whenever a
* run-time error occures a window will be displayed showing the context
* of the error and giving the user the option of continuing,aborting, etc.*
*****)
```

```
DEFINITION MODULE RunTimeErrors;
(*$L+*)
```

```
PROCEDURE InstallErrorHandler;
(* Install run-time error processor. *)
```

```
PROCEDURE RemoveErrorHandler;
(* Remove run-time error processor. *)
```

```
END RunTimeErrors.
```

Module Semaphores

```

(*****
*
*          Modula-2 Software Construction Set
*          (c) 1986 by Leon Frenkel
*
*****
* Name: Semaphores.DEF          Version: Amiga.00.00
* Created: 11/19/86   Updated: 11/22/86   Author: Leon Frenkel
* Description: Exec Semaphores Types/Functions.
*****)

DEFINITION MODULE Semaphores;

FROM SYSTEM IMPORT ADDRESS;
FROM Lists IMPORT MinList;
FROM Nodes IMPORT Node, MinNode;
FROM Ports IMPORT MsgPort, Message, MessagePtr;
FROM Tasks IMPORT TaskPtr;

TYPE
  SemaphorePtr = POINTER TO Semaphore;
  Semaphore = RECORD
    smMsgPort : MsgPort;
    smBids    : INTEGER;
  END;

  (* SignalSemaphore *)
  (* This is the structure used to request a signal semaphore *)
  SemaphoreRequestPtr = POINTER TO SemaphoreRequest;
  SemaphoreRequest = RECORD
    srLink : MinNode;
    srWaiter : TaskPtr;
  END;

  (* This is the actual semaphore itself *)
  SignalSemaphorePtr = POINTER TO SignalSemaphore;
  SignalSemaphore = RECORD
    ssLink : Node;
    ssNestCount : INTEGER;
    ssWaitQueue : MinList;
    ssMultipleLink : SemaphoreRequest;
    ssOwner : TaskPtr;
    ssQueueCount : INTEGER;
  END;

PROCEDURE AddSemaphore(VAR signalSemaphore: SignalSemaphore);
(* Add a signal semaphore to the system.

   signalSemaphore - an initialized signal semaphore structure *)

PROCEDURE AttemptSemaphore(VAR signalSemaphore: SignalSemaphore): BOOLEAN;
(* Try to obtain without blocking.

   signalSemaphore - an initialized signal semaphore structure

   result - TRUE if the semaphore was lock, FALSE if some other task
            already possessed the semaphore *)

PROCEDURE FindSemaphore(name: ADDRESS): SignalSemaphorePtr;
(* Find a given system signal semaphore.

   name - name of the semaphore to find

   result - a pointer to the signal semaphore, or NIL if not found *)

PROCEDURE InitSemaphore(VAR signalSemaphore: SignalSemaphore);
(* Initialize a signal semaphore.

   signalSemaphore - an uninitialized signal semaphore structure *)

```

Module Semaphores

```
PROCEDURE ObtainSemaphore(VAR signalSemaphore: SignalSemaphore);  
(* Gain exclusive access to a semaphore.
```

signalSemaphore - an initialized signal semaphore structure *)

```
PROCEDURE ObtainSemaphoreList(VAR list: SignalSemaphore);  
(* Get a list of semaphores.
```

list - a list of signal semaphores *)

```
PROCEDURE Procure(VAR semaphore: Semaphore; VAR bidMessage: Message): BOOLEAN;  
(* Bid for a message lock (semaphore).
```

semaphore - a semaphore message port
bidMessage - a message

result - TRUE when the semaphore is free. In such cases no waiting needs
to be done. If FALSE, the task should wait at its bidMessage
reply port *)

```
PROCEDURE ReleaseSemaphore(VAR signalSemaphore: SignalSemaphore);  
(* Make a signal semaphore available to others.
```

signalSemaphore - an initialized signal semaphore structure *)

```
PROCEDURE ReleaseSemaphoreList(VAR list: SignalSemaphore);  
(* Make a list of semaphore available.
```

list - a list of signal semaphores *)

```
PROCEDURE RemSemaphore(VAR signalSemaphore: SignalSemaphore);  
(* Remove a signal semaphore from the system.
```

signalSemaphore - an initialized signal semaphore structure *)

```
PROCEDURE Vacate(VAR semaphore: Semaphore);  
(* Release a message lock (semaphore).
```

semaphore - the semaphore message port representing the semaphore to be
freed *)

END Semaphores.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
* Name: SerialDevice.DEF                      *
* Created: 11/27/86   Updated: 12/31/86   Author: Leon Frenkel *
* Description: "serial.device" types.          *
*****)

```

```

DEFINITION MODULE SerialDevice;

```

```

FROM SYSTEM IMPORT BYTE;
FROM IODevices IMPORT CmdNonStd, IOStdReq;

```

```

CONST
  SerialName = "serial.device";

  SDCmdQuery      = CmdNonStd + 0;
  SDCmdBreak      = CmdNonStd + 1;
  SDCmdSetParams  = CmdNonStd + 2;

```

```

TYPE
  (* array of termination char's *)
  IOTArrayPtr = POINTER TO IOTArray;
  IOTArray = RECORD
    TermArray0 : LONGCARD;
    TermArray1 : LONGCARD;
  END;

```

```

CONST
  (* IOStdRequest.ioFlags := IOFlagsSet{ } *)
  IOSerBufRead      = 7; (* from read buffer bit *)
  IOSerQueued       = 6; (* rqst-queued bit *)
  IOSerAbort        = 5; (* rqst-aborted bit *)
  IOSerActive       = 4; (* rqst-queued-or-current bit *)

```

```

TYPE
  SerFlags = (SerParityOn, (* parity-enabled bit *)
              SerParityOdd, (* parity feature enabled bit *)
              Ser7Wire, (* RS232 7-wire protocol *)
              SerQueuedBrk, (* queue this Break ioRqst *)
              SerRadBoogie, (* high-speed mode active bit *)
              SerShared, (* non-exclusive access bit *)
              SerEOFMode, (* EOF mode enabled bit *)
              SerXDisabled); (* xOn-xOff feature disabled bit *)
  SerFlagsSet = SET OF SerFlags;

```

```

  SerExtFlags = (SExtMark, (* if mark=space, use mark bit *)
                 SExtMSPOn, (* mark-space parity bit *)
                 SEF2, SEF3, SEF4, SEF5, SEF6, SEF7,
                 SEF8, SEF9, SEF10, SEF11, SEF12, SEF13,
                 SEF14, SEF15, SEF16, SEF17, SEF18, SEF19,
                 SEF20, SEF21, SEF22, SEF23, SEF24, SEF25,
                 SEF26, SEF27, SEF28, SEF29, SEF30, SEF31);
  SerExtFlagsSet = SET OF SerExtFlags;

```

```

  SerStatus = (IOSTOverRun, (* status work RBF overrun bit *)
               IOSTWroteBreak, (* break was latest output bit *)
               IOSTReadBreak, (* break was latest input bit *)
               IOSTXOffWrite, (* transmit currently xOFF'ed bit *)
               IOSTXOffRead, (* receive currently xOFF'ed bit *)
               IOST5, IOST6, IOST7, IOST8, IOST9, IOST10,
               IOST11, IOST12, IOST13, IOST14, IOST15);
  SerStatusSet = SET OF SerStatus;

```

Module SerialDevice

```

(*****
(* CAUTION !! IF YOU ACCESS the serial.device, you MUST (!!!!) use an *)
(* IOExtSer-sized structure or you may overlay innocent memory !! *)
(*****

IOExtSerPtr = POINTER TO IOExtSer;
IOExtSer = RECORD
    IOSer      : IOStdReq;
    ioCtlChar  : LONGCARD; (* control char(order=xON,xOFF,INQ,ACK)*)
    ioRBufLen  : LONGCARD; (* length in bytes of serial read buf *)
    ioExtFlags : SerExtFlagsSet; (* additional flags (see above) *)
    ioBaud     : LONGCARD; (* baud rate requesterd (true baud) *)
    ioBrkTime  : LONGCARD; (* duration of break signal in microsec*)
    ioTermArray : IOTArray; (* termination character array *)
    ioReadLen  : BYTE; (* bits per read char (# of bits) *)
    ioWriteLen : BYTE; (* bits per write char (# of bits) *)
    ioStopBits : BYTE; (* stopbits for read (# of bits) *)
    ioSerFlags : SerFlagsSet; (* SerFlags bit defs above *)
    ioStatus   : SerStatusSet;
END;

(* status of serial port, as follows:
*
*      BIT  ACTIVE  FUNCTION
*      0    low    busy
*      1    low    paper out
*      2    low    select
*      3    low    Data Set Ready
*      4    low    Clear To Send
*      5    low    Carrier Detect
*      6    low    Ready To Send
*      7    low    Data Terminal Ready
*      8    high   read overrun
*      9    high   break sent
*     10    high   break received
*     11    high   transmit x-OFFed
*     12    high   receive x-OFFed
*     13-15 reserved
*)

CONST
    SerErrDevBusy      = 1;
    SerErrBaudMismatch = 2;
    SerErrInvBaud       = 3;
    SerErrBufErr        = 4;
    SerErrInvParam      = 5;
    SerErrLineErr       = 6;
    SerErrNotOpen       = 7;
    SerErrPortReset     = 8;
    SerErrParityErr     = 9;
    SerErrInitErr       = 10;
    SerErrTimerErr      = 11;
    SerErrBufOverflow   = 12;
    SerErrNoDSR         = 13;
    SerErrNoCTS         = 14;
    SerErrDetectedBreak = 15;

END SerialDevice.

```



```

(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: Sprites.DEF                               Version: Amiga.00.00                       *
* Created: 11/20/86   Updated: 11/23/86   Author: Leon Frenkel                             *
* Description: Graphics Sprites types/functions.                                           *
*****)

```

```

DEFINITION MODULE Sprites;

```

```

FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Views IMPORT ViewPort;

```

```

CONST

```

```

  SpriteAttached = 80H;

```

```

  (* Pass as parameter to GetSprite() if no sprite number preference *)
  AnySprite = -1;

```

```

TYPE

```

```

  SimpleSpritePtr = POINTER TO SimpleSprite;
  SimpleSprite = RECORD
    posctldata : ADDRESS;
    height     : CARDINAL;
    x          : CARDINAL; (* current x position *)
    y          : CARDINAL; (* current y position *)
    num        : CARDINAL;
  END;

```

```

PROCEDURE ChangeSprite(VAR vp: ViewPort; VAR s: SimpleSprite; newdata: ADDRESS);
(* Change the sprite image pointer.

```

```

  vp - ViewPort structure that this sprite is relative to or NIL if relative
  only top of View

```

```

  s - SimpleSprite structure

```

```

  newdata - pointer to data structure of the following form:

```

```

    spriteimage = RECORD
      posctl : ARRAY [0..1] OF CARDINAL;
      data   : ARRAY [1..height][0..1] OF CARDINAL;
      reserved : ARRAY [0..1] OF CARDINAL;
    END;

```

```

  Programmer must initialize reserved[2]. Spriteimage must be
  in CHIP memory. The height subfield of the SimpleSprite structure
  must be set to reflect the height of the new spriteimage BEFORE
  calling ChangeSprite. The programmer may allocate two sprites to
  handle a single attached sprite. After GetSprite, ChangeSprite,
  the programmer can set the SPRITE_ATTACHED bit in posctl[1] of the
  odd numbered sprite. If you need more than 8 sprites look up VSprites
  in the graphics documentation. *)

```

```

PROCEDURE FreeSprite(pick: INTEGER);

```

```

(* Return sprite for use by other and virtual sprite machine.

```

```

  pick - number in range of 0..7 *)

```

```

PROCEDURE GetSprite(VAR sprite: SimpleSprite; pick: INTEGER): INTEGER;

```

```

(* Attempt to get a sprite for the simple sprite manager.

```

```

  sprite - programmers SimpleSprite structure

```

```

  pick - number in the range of 0..7 or -1 for next sprite

```

```

  result - allocated sprite number, or -1 if sprite could not be gotten *)

```

Module Sprites

```
PROCEDURE MoveSprite(VAR vp: ViewPort; VAR sprite: SimpleSprite; x, y: INTEGER);
(* Move sprite to a pointer relative to top of viewport.
```

```
    vp - pointer to ViewPort structure
    sprite - pointer to SimpleSprite structure
    x - new x position relative to top of viewport or view
    y - new y position relative to top of viewport or view *)
```

END Sprites.

```

*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
* Name: Storage.DEF                      Version: Amiga.00.00 *
* Created: 12/03/86   Updated: 01/20/87   Author: Leon Frenkel *
* Description: Memory management module as defined in appendix 2 of *
* "Programming in Modula-2" by N. Wirth. This module is included to *
* allow programs written on other systems to be ported and vice versa, *
* to access the full memory management capabilities of the Amiga use the *
* module "Memory". *
*****

```

```
DEFINITION MODULE Storage;
```

```
FROM SYSTEM IMPORT ADDRESS;
```

```
(* $L+ *)
```

```
PROCEDURE ALLOCATE(VAR a: ADDRESS; size: LONGCARD);
```

```
(* Allocate an area of given size and returns its address in a.
```

```

  a - variable to be set to pointer to memory, set to NIL if not enough mem
  size - number of bytes to allocate *)

```

```
PROCEDURE DEALLOCATE(VAR a: ADDRESS; size: LONGCARD);
```

```
(* Frees the area at address a with the given size.
```

```

  a - variable which contains the adr of the memory to be deallocated
  size - number of bytes to free *)

```

```
PROCEDURE Available(size: LONGCARD): BOOLEAN;
```

```

(* Available returns TRUE if size bytes could be allocated. Since the Amiga
  is a multi-tasking system, the only way to be sure that the memory will
  actually be available when the ALLOCATE is issued is to use Forbid() and
  Permit() around the call to this procedure and the call to the ALLOCATE
  procedure which may follow it.

```

```

  size - number of bytes we would require

```

```

  result - TRUE if the required amount of memory is available *)

```

```
END Storage.
```

Module Strings

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Strings.DEF                      Version: Amiga.00.00
* Created: 12/03/86   Updated: 02/09/87   Author: Leon Frenkel
* Description: Functions for handling null-terminated strings, defined
* as arrays of chars. This implementation can handle strings of up to
* 32767 chars.
*****)
```

DEFINITION MODULE Strings;

(* \$L+ *)

TYPE

(* Result type of CompareString() and CompareStringCAP() *)
Relation = (less, equal, greater);

PROCEDURE StringLength(string: ARRAY OF CHAR): CARDINAL;

(* returns the number of characters in the string, up to but excluding the string terminator.

string - string whose length is to be measured

result - length of string in chars *)

PROCEDURE ConvStringToUpperCase(VAR string: ARRAY OF CHAR);

(* Force the chars in the string to upper case. Only lower case letters are effected, all other chars remain the same.

string - string to be converted to upper case *)

PROCEDURE CompareString(string1, string2: ARRAY OF CHAR): Relation;

(* return less, equal or greater depending on the lexical order of the strings.

string1 - the string on the left side of the compare

string2 - the string on the right side of the compare

result - returns the result of comparing the strings: <,<=,> *)

PROCEDURE CompareStringCAP(string1, string2: ARRAY OF CHAR): Relation;

(* return less, equal or greater depending on the lexical order of the strings. Both strings are converted to upper case before comparing!

string1 - the string on the left side of the compare

string2 - the string on the right side of the compare

result - returns the result of comparing the strings: <,<=,> *)

PROCEDURE CopyString(VAR toString: ARRAY OF CHAR; fromString: ARRAY OF CHAR);

(* Copies the string "fromString" to the string "toString". The string is truncated if necessary.

toString - the destination string for the copy

fromString - the source string for the copy *)

PROCEDURE ConcatString(VAR toString: ARRAY OF CHAR; fromString: ARRAY OF CHAR);

(* Append the "fromString" to the end of the "toString". If the toString is not large enough to accomodate the new string it will be truncated.

toString - Destination string for the concatenation

fromString - string to be appended to the "toString" *)

```

PROCEDURE InsertSubString(VAR baseString: ARRAY OF CHAR;
                          subString: ARRAY OF CHAR; start: CARDINAL);
(* Inserts "subString" into "baseString": inserting starts after "start"
characters of "baseString" are skipped; the rest of "baseString" is
shifted up and truncated as necessary. If start > StringLength(baseString)
the result is undefined

```

```

baseString - string into which to insert
subString - the string to be inserted
start - the baseString char after which to begin insert *)

```

```

PROCEDURE OverwriteWithSubString(VAR baseString: ARRAY OF CHAR;
                                subString: ARRAY OF CHAR; start: CARDINAL);
(* Like insert, but does not shift up part of "baseString": "baseString" is
lengthened and "subString" truncated as necessary.

```

```

baseString - the string to be partially overwritten
subString - the string to overwrite with
start - character after which to begin storing the subString *)

```

```

PROCEDURE DeleteSubString(VAR baseString: ARRAY OF CHAR;
                          fromPosition: CARDINAL; length: CARDINAL);
(* Deletes "length" characters in "baseString" starting at "fromPosition";
the procedure shifts down any character of "baseString" following the
deletion.

```

```

baseString - string from which to delete characters
fromPosition - the char position where to begin delete
length - the number of chars to delete *)

```

```

PROCEDURE ExtractSubString(VAR subString: ARRAY OF CHAR;
                           baseString: ARRAY OF CHAR;
                           fromPosition, length: CARDINAL);
(* returns in subString the characters (truncated as necessary) of
"baseString" from "fromPosition" through "fromPosition" + "length" - 1.

```

```

subString - the string into which the extracted string is to be stored
baseString - the string from which to extract the sub string
fromPosition - begin extraction from this position
length - number of chars to be extracted *)

```

```

PROCEDURE LocateSubString(baseString, subString: ARRAY OF CHAR;
                          start, end: CARDINAL): INTEGER;
(* Searches for "subString" in positions "start" through "end" of
"baseString".

```

```

baseString - string to be searched for sub-string
subString - string for which to search in baseString
start - starting position for search
end - ending position for search

```

```

result - position of string, or -1 if not found *)

```

```

PROCEDURE LocateChar(string: ARRAY OF CHAR; c: CHAR;
                    start, end: CARDINAL): INTEGER;
(* Searches for "c" in positions "start" through "end" of string.

```

```

string - string to be searched for char
c - char to be searched for
start - starting position of search in string
end - ending position of search for string

```

```

result - position of char, or -1 if not found *)

```

```

END Strings.

```

Module System

```
(*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: System.DEF                               Version: Amiga.00.00                       *
* Created: 11/17/86   Updated: 07/24/87   Author: Leon Frenkel                             *
* Description: AmigaDOS Modula-2 program startup code and run-time support                 *
* procedures.                                                                              *
*****)
```

DEFINITION MODULE System;

FROM SYSTEM IMPORT ADDRESS;

VAR

(*****
(* The following libraries are opened and closed by the startup code! *)
*****)

ExecBase : ADDRESS; (* "exec.library" *)
DOSBase : ADDRESS; (* "dos.library" *)
MathBase : ADDRESS; (* "mathffp.library" *)
IntuitionBase : ADDRESS; (* "intuition.library" *)
GfxBase : ADDRESS; (* "graphics.library" *)
LayersBase : ADDRESS; (* "layers.library" *)

CmdLinePtr : ADDRESS; (* Ptr to command line arguments string *)
CmdLineLength : LONGCARD; (* Length of command line arguments string *)

StackPtr : ADDRESS; (* Stack Ptr Reg (A7) on entry into program *)
StackSize : LONGCARD; (* Size of stack allocated for program *)

StdInput : ADDRESS; (* file handle for standard input *)
StdOutput : ADDRESS; (* file handle for standard output *)

(* argc - number of parsed arguments *)
(* argv - pointer to an array of string pointers to actual arguments *)
(* if argc = 0 THEN WBenchMsg = ptr to the message sent to us by WB *)
argc : CARDINAL;
argv : POINTER TO ARRAY [0..255] OF POINTER TO ARRAY [0..255] OF CHAR;
WBenchMsg : ADDRESS;

(* Code to be returned on exit from program, defaults to 0 *)
CLIReturnCode : LONGINT;

(* === Modula-2 Run-Time Support Procedures (Internal use only!) === *)

PROCEDURE HALTX;
(* Terminate program returning to the CLI or to the Workbench. *)

PROCEDURE MULU32;
(* Multiply two unsigned 32-bit numbers. *)

PROCEDURE DIVU32;
(* Divide two unsigned 32-bit numbers. *)

PROCEDURE MULS32;
(* Multiply two signed 32-bit numbers. *)

PROCEDURE DIVS32;
(* Divide two signed 32-bit numbers. *)

END System;


```

(* === Single-Precision Floating Point Math Functions === *)

PROCEDURE FADDs(adder, addend: REAL): REAL;
(* Add two floating point numbers. *)

PROCEDURE FSUBs(minuend, subtrahend: REAL): REAL;
(* Subtract two floating point numbers. *)

PROCEDURE FMULs(multiplicand, multiplier: REAL): REAL;
(* Multiply two floating point numbers. *)

PROCEDURE FDIVs(dividend, divisor: REAL): REAL;
(* Divide two floating point numbers. *)

PROCEDURE FREMs(dividend, divisor: REAL): REAL;
(* Take the remained of the division of two floating point numbers. *)

PROCEDURE FCMPs(first, second: REAL);
(* Compare two floating point numbers. Sets the Cond Codes. *)

PROCEDURE FNEGs(toNeg: REAL): REAL;
(* Negate a floating point number. *)

PROCEDURE FABSS(toAbs: REAL): REAL;
(* Take the absolute value of a floating point number. *)

PROCEDURE FLOATs(toFloat: LONGINT): REAL;
(* Convert a LONGINT to a REAL. *)

PROCEDURE TRUNCs(toTrunc: REAL): LONGINT;
(* Convert a REAL to a LONGINT. *)

(* === Double-Precision Floating Point Math Functions === *)

PROCEDURE FADDd(adder, addend: LONGREAL): LONGREAL;
(* Add two floating point numbers. *)

PROCEDURE FSUBd(minuend, subtrahend: LONGREAL): LONGREAL;
(* Subtract two floating point numbers. *)

PROCEDURE FMULD(multiplicand, multiplier: LONGREAL): LONGREAL;
(* Multiply two floating point numbers. *)

PROCEDURE FDIVd(dividend, divisor: LONGREAL): LONGREAL;
(* Divide two floating point numbers. *)

PROCEDURE FREMd(dividend, divisor: LONGREAL): LONGREAL;
(* Take the remained of the division of two floating point numbers. *)

PROCEDURE FCMPd(first, second: LONGREAL);
(* Compare two floating point numbers. Sets the Cond Codes. *)

PROCEDURE FNEGd(toNeg: LONGREAL): LONGREAL;
(* Negate a floating point number. *)

PROCEDURE FABSD(toAbs: LONGREAL): LONGREAL;
(* Take the absolute value of a floating point number. *)

PROCEDURE FLOATd(toFloat: LONGINT): LONGREAL;
(* Convert a LONGINT to a REAL. *)

PROCEDURE TRUNCd(toTrunc: LONGREAL): LONGINT;
(* Convert a REAL to a LONGINT. *)

PROCEDURE FLONG(toConvert: REAL): LONGREAL;
(* Convert a single precision floating point number to double precision. *)

PROCEDURE FSHORT(toConvert: LONGREAL): REAL;
(* Convert a double precision floating point number to single precision. *)

END System.

```

Module Tasks

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Tasks.DEF                      Version: Amiga.00.00
* Created: 11/16/86   Updated: 05/08/87   Author: Leon Frenkel
* Description: Exec Tasks Types/Functions.
*****)

```

DEFINITION MODULE Tasks;

```

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM Nodes IMPORT Node;
FROM Lists IMPORT List;

```

CONST

```

CurrentTask = NIL; (* Passed to FindTask or RemTask to indicate current task*)

```

```

AnySignal = -1; (* Passed to AllocSignal to return any signal *)
NoSignals = -1; (* Returned by AllocSignal to indicate no signals *)
MaxSignal = 31; (* Highest signal number *)
AnyTrap = -1; (* Passed to AllocTrap to return any trap *)
NoTraps = -1; (* Returned by AllocTrap to indicate no traps *)
MaxTrap = 15; (* Highest trap number *)

```

(* Predefined signal bits *)

```

SigAbort = 0;
SigChild = 1;
SigBlit = 4;
SigSingle = 4;
SigDos = 8;

```

TYPE

```

SignalRange = [AnySignal..MaxSignal];
SignalSet = SET OF [0..MaxSignal];

```

```

TrapRange = [AnyTrap..MaxTrap];
TrapSet = SET OF [0..MaxTrap];

```

```

TaskFlags = (TProcTime, T1, T2, T3, TStackChk, TExcept, TSwitch,
             TLaunch);

```

```

TaskFlagsSet = SET OF TaskFlags;

```

```

TaskState = (TSInvalid, TSAdded, TSTRun, TSReady, TSWait, TSExcept, TSRemoved);
TaskStateSet = SET OF TaskState;

```

TaskPtr = POINTER TO Task;

Task = RECORD

```

  tcNode      : Node;
  tcFlags     : TaskFlagsSet;
  tcState     : TaskStateSet;
  tcIDNestCnt : BYTE;      (* intr disabled nesting *)
  tcTDNestCnt : BYTE;      (* task disabled nesting *)
  tcSigAlloc  : SignalSet; (* signals allocated *)
  tcSigWait   : SignalSet; (* signals we are waiting for *)
  tcSigRecvd  : SignalSet; (* signals we have received *)
  tcSigExcept : SignalSet; (* signals we take excepts for *)
  tcTrapAlloc : TrapSet;   (* traps allocated *)
  tcTrapAble  : TrapSet;   (* traps enabled *)
  tcExceptData : ADDRESS;   (* points to except data *)
  tcExceptCode : ADDRESS;   (* points to except code *)
  tcTrapData  : ADDRESS;   (* points to trap code *)
  tcTrapCode  : ADDRESS;   (* points to trap data *)
  tcSPReg     : ADDRESS;   (* stack pointer *)
  tcSPLower   : ADDRESS;   (* stack lower bound *)
  tcSPUpper   : ADDRESS;   (* stack upper bound + 2 *)
  tcSwitch    : ADDRESS;   (* task losing CPU *)
  tcLaunch    : ADDRESS;   (* task getting CPU *)
  tcMemEntry  : List;      (* allocated memory *)
  tcUserData  : ADDRESS;   (* per task data *)

```

END;

PROCEDURE AddTask(VAR task: Task; initialPC, finalPC: ADDRESS);
 (* Add a task to the system.

task - the task control block
 initialPC - the initial entry point's address
 finalPC - the finalization code entry point's address. If NIL then the
 system will use a general finalizer *)

PROCEDURE AllocSignal(signalNum: SignalRange): SignalRange;

(* Allocate a signal bit. WARNING: Signals may not be allocated or freed from
 exception handling code.

signalNum - the desired signal number 0..31 or -1 for no preference

result - the signal bit number allocated 0..31, if no signals are available
 this function returns -1 *)

PROCEDURE AllocTrap(trapNum: TrapRange): TrapRange;

(* Allocate a processor trap vector. WARNING: Traps may not be allocated or
 freed from exception handling code.

trapNum - the desired trap number 0..15 or -1 for no preferences

result - the trap number allocated 0..15 or -1 if no trap is available *)

PROCEDURE FindTask(name: ADDRESS): TaskPtr;

(* Find a task with the given name or find oneself.

name - pointer to a name string or NIL to find current task

result - pointer to the task or NIL if not found *)

PROCEDURE FreeSignal(signalNum: SignalRange);

(* Free a signal bit. WARNING: Signals may not be allocated or freed from
 exception handling code.

signalNum - the signal number to free *)

PROCEDURE FreeTrap(trapNum: TrapRange);

(* Free a processor trap. WARNING: Traps may not be allocated or freed from
 exception handling code.

trapNum - the trap number to free 0..15 *)

PROCEDURE RemTask(task: TaskPtr);

(* Remove a task from the system.

task - pointer to the task node representing the task to be removed. A NIL
 value indicates self removal *)

PROCEDURE SetExcept(newSignals, signalMask: SignalSet): SignalSet;

(* Define certain signals to cause exceptions.

newSignals - the new values for the signals specified in signalMask
 signalMask - the set of signals to be effected

result - the prior exception signals

EXAMPLE

SetExcept(SignalSet{}, SignalSet{}) - will return the current state of
 all exception signals

SetExcept(SignalSet{4,5,9,12}, SignalSet{4,9,12}) - change a few
 exception signals *)

Module Tasks

```
PROCEDURE SetSignal(newSignals, signalMask: SignalSet): SignalSet;  
(* Define the state of this task's signals.
```

newSignals - the new values for the signals specified in signalSet
signalMask - the set of signals to be effected

result - the prior values for all signals *)

```
PROCEDURE SetTaskPri(task: Task; priority: INTEGER): INTEGER;  
(* Get and set the priority of a task.
```

task - task to be affected
priority - the new priority for the task -128..127

result - the tasks previous priority *)

```
PROCEDURE Signal(task: Task; signals: SignalSet);  
(* Signal a task.
```

task - the task to be signalled
signals - the signals to be sent *)

```
PROCEDURE Wait(signalSet: SignalSet): SignalSet;  
(* Wait for one or more signals.
```

signalSet - the set of signals for which to wait

result - the set of signals which caused the return from wait *)

END Tasks.

```

(*****
*
*      Modula-2 Software Construction Set
*      (c) 1986 by Leon Frenkel
*
*****
* Name: TasksUtil.DEF      Version: Amiga.00.00
* Created: 01/04/87   Updated: 01/14/87   Author: Leon Frenkel
* Description: Exec Support Procedures for Tasks.
*****)

```

```
DEFINITION MODULE TasksUtil;
```

```
FROM SYSTEM IMPORT ADDRESS;
FROM Tasks IMPORT Task, TaskPtr;
```

```
(* $L+*)
```

```
PROCEDURE CreateTask(name: ADDRESS; pri: INTEGER; initPC: ADDRESS;
                    stackSize: LONGCARD): TaskPtr;
```

```
(* Create a task.
```

```
name - a pointer to null-term string
```

```
pri - priority of task
```

```
initPC - ptr to be passed to AddTask as the initialPC parameter
```

```
stackSize - used in the setup of tcSPLower and tcSPUpper
```

```
result - returns a pointer to the new Task, or NIL if the request failed *)
```

```
PROCEDURE DeleteTask(task: TaskPtr);
```

```
(* Frees up a tasks' stuff as allocated by CreateTask().
```

```
task - pointer to the task node representing the task to be removed. A NIL
value indicates self removal *)
```

```
END TasksUtil.
```

Module Terminal

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Terminal.DEF                      Version: Amiga.00.00
* Created: 11/19/86   Updated: 01/20/87   Author: Leon Frenkel
* Description: Standard Terminal I/O as described in Programming in
* appendix 2 of Programming in Modula-2 by N. Wirth.
* The procedures in this module read/write to the I/O files initially
* assigned to a program. This is usually the window from which the program
* started but may be redirected by the user to any file using the
* AmigaDOS ">" and "<" after the name of the program.
*****)
```

DEFINITION MODULE Terminal;

(* \$L+ *)

VAR

(* Result of previous Read(c) procedure: TRUE = end of file, FALSE = not eof *)
eof : BOOLEAN;

PROCEDURE Read(VAR ch: CHAR);

(* Read a character from the terminal, if a character is not ready then
wait till it is typed. *)

PROCEDURE BusyRead(VAR ch: CHAR);

(* Returns OC if no character was typed. *)

PROCEDURE ReadAgain;

(* Causes the last character read to be returned again upon the next of Read. *)

PROCEDURE Write(ch: CHAR);

(* Output a character to the terminal. *)

PROCEDURE WriteLn;

(* Terminate line. *)

PROCEDURE WriteString(s: ARRAY OF CHAR);

(* Output a string of characters to the terminal. *)

END Terminal.

```

(*****
*                               Modula-2 Software Construction Set
*                               (c) 1986 by Leon Frenkel
*                               *****)
* Name: TermInOut.DEF                               Version: Amiga.00.00
* Created: 08/07/87   Updated: 08/07/87   Author: Leon Frenkel
* Description: A very small version of InOut for output to terminal ONLY!
*****)

```

```
DEFINITION MODULE TermInOut;
```

```
(* $L+*)
```

```

CONST EOL = 12C;
VAR Done: BOOLEAN;
    termCH: CHAR; (*terminating character in ReadInt, ReadCard*)

```

```

(* TRUE = echo chars in ReadString, FALSE = don't echo *)
Echo: BOOLEAN; (* Default: TRUE *)

```

```

PROCEDURE Read(VAR ch: CHAR);
(*Done := NOT in.eof*)

```

```

PROCEDURE ReadString(VAR s: ARRAY OF CHAR);
(*read string, i.e. sequence of characters not containing
 blanks nor control characters; leading blanks are ignored.
 Input is terminated by any character <= " ";
 this character is assigned to termCH.
 DEL is used for backspacing when input from terminal*)

```

```

PROCEDURE ReadInt(VAR x: INTEGER);
(*read string and convert to integer. Syntax:
 integer = ["+"|-"] digit {digit}.
 Leading blanks are ignored.
 Done := "integer was read"*)

```

```

PROCEDURE ReadCard(VAR x: CARDINAL);
(*read string and convert to cardinal. Syntax:
 cardinal = digit {digit}.
 Leading blanks are ignored.
 Done := "cardinal was read"*)

```

```
PROCEDURE Write(ch: CHAR);
```

```
PROCEDURE WriteLn; (*terminate line*)
```

```
PROCEDURE WriteString(s: ARRAY OF CHAR);
```

```

PROCEDURE WriteInt(x: INTEGER; n: CARDINAL);
(*write integer x with (at least) n characters on file "out".
 If n is greater than the number of digits needed,
 blanks are added preceding the number*)

```

```
PROCEDURE WriteCard(x, n: CARDINAL);
```

```
PROCEDURE WriteOct(x, n: CARDINAL);
```

```
PROCEDURE WriteHex(x, n: CARDINAL);
```

```
END TermInOut.
```

Module Text

```

*****
*                               Modula-2 Software Construction Set                               *
*                               (c) 1986 by Leon Frenkel                                       *
*****
* Name: Text.DEF                               Version: Amiga.00.00                          *
* Created: 11/20/86   Updated: 05/08/87   Author: Leon Frenkel                             *
* Description: Graphics Text Types/functions.                                             *
*****

```

DEFINITION MODULE Text;

```

FROM SYSTEM IMPORT BYTE, ADDRESS;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Ports IMPORT Message;
FROM Rasters IMPORT RastPort;

```

```

TYPE
  (* Font Styles *)
  FontStyle = (Underlined, (* underlined (under baseline) *)
              Bold,        (* bold face text (ORED w/ shifted) *)
              Italic,      (* italic (slanted 1:2 right) *)
              Extended);   (* extended face (wider than normal) *)
  FontStyleSet = SET OF FontStyle;

```

```

CONST
  (* normal text (no style bits set) *)
  NormalFontStyle = FontStyleSet{};

```

```

TYPE
  (* Font Flags *)
  FontFlags = (RomFont,      (* font is in rom *)
              DiskFont,     (* font is from diskfont.library *)
              RevPath,      (* designed path is reversed (e.g. left) *)
              TallDot,      (* designed for hires non-interlaced *)
              WideDot,      (* designed for lores interlaced *)
              Proportional, (* character sizes can vary from nominal *)
              Designed,     (* size is "designed", no constructed *)
              Removed);     (* the font has been removed *)
  FontFlagsSet = SET OF FontFlags;

```

```

TextAttrPtr = POINTER TO TextAttr;
TextAttr = RECORD
  taName : ADDRESS;      (* name of the font *)
  taYSize : CARDINAL;    (* height of the font *)
  taStyle : FontStyleSet; (* intrinsic font style *)
  taFlags : FontFlagsSet; (* font preferences and flags *)
END;

```

```

(* TextFonts node *)
TextFontPtr = POINTER TO TextFont;
TextFont = RECORD
  tfMessage : Message;      (* reply message for font removal *)
  tfYSize : CARDINAL;      (* font height *)
  tfStyle : FontStyleSet;   (* font style *)
  tfFlags : FontFlagsSet;   (* preferences and flags *)
  tfXSize : CARDINAL;      (* nominal font width *)
  tfBaseline : CARDINAL;    (* distance from the top of char to baseline *)
  tfBoldSmear : CARDINAL;   (* smear to affect a bold enhancement *)
  tfAccessors : CARDINAL;   (* access count *)
  tfLoChar : CHAR;          (* the first char described here *)
  tfHiChar : CHAR;          (* the last char described here *)
  tfCharData : ADDRESS;     (* the bit character data *)
  tfModulo : CARDINAL;      (* the row modulo for the strike font data *)
  tfCharLoc : ADDRESS;      (* ptr to location data for the strike font *)
  (* 2 words: bit offset then size *)
  tfCharSpace : ADDRESS;    (* ptr to words of proportional spacing data *)
  tfCharKern : ADDRESS;     (* ptr to words of kerning data *)
END;

```

```

PROCEDURE AddFont(VAR textFont: TextFont);
(* Add a font to the system list.

```

textFont - a TextFont structure in public ram *)

```

PROCEDURE AskFont(VAR rp: RastPort; VAR textAttr: TextAttr);
(* Get the text attributes of the current font.

rp - the RastPort from which the text attributes are extracted
textAttr - the TextAttr structure to be filled *)

PROCEDURE AskSoftStyle(VAR rp: RastPort): FontStyleSet;
(* Get the soft style bits of the current font.

rp - the RastPort from which the font and style are extracted
result - those bits in the style that are algorithmically generated *)

PROCEDURE CloseFont(VAR font: TextFont);
(* Release a pointer to a system font.

font - a font as returned by OpenFont or OpenDiskFont *)

PROCEDURE OpenFont(VAR textAttr: TextAttr): TextFontPtr;
(* Get a pointer to a system font.

textAttr - a TextAttr structure that describes the text font attributes desired
result - pointer to font, or NIL if font not found *)

PROCEDURE RemFont(VAR textFont: TextFont);
(* Remove a font from the system list.

textFont - the TextFont structure to remove *)

PROCEDURE SetFont(VAR rp: RastPort; VAR font: TextFont);
(* Set the text font and attributes in a RastPort.

rp - the RastPort in which the text attributes are to be changed
font - a TextFont structure returned from OpenFont or OpenDiskFont *)

PROCEDURE SetSoftStyle(VAR rp: RastPort; style: FontStyleSet;
enable: FontStyleSet): FontStyleSet;
(* Set the soft style of the current font.

rp - the RastPort from which the font and style are extracted
style - the new font style to set, subject to enable
enable - those bits in style to be changed

result - the resulting style, both as a result of previous soft style
selection, the of this function, and the style inherent in the font
*)

PROCEDURE Text(VAR rp: RastPort; string: ADDRESS; count: INTEGER);
(* Write text characters (no formatting).

rp - a RastPort which describes where the text is to be output
string - the string to be output
count - the length of the string, if zero then no characters to output *)

PROCEDURE TextLength(VAR rp: RastPort; string: ADDRESS; count: INTEGER): INTEGER;
(* Determine raster length of text data.

rp - a RastPort which describes where the text is to be output
string - the string for which to determine the length
count - the length of the string, if zero then no characters to output

result - the number of pixels in x this text would occupy *)

END Text.

```


Module TimerDevice

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*                               *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****)
* Name: TimerDevice.DEF                      Version: Amiga.00.00
* Created: 11/26/86   Updated: 01/17/87   Author: Leon Frenkel
* Description: Timer Device.
*****)

DEFINITION MODULE TimerDevice;

FROM SYSTEM IMPORT ADDRESS;
FROM IODevices IMPORT CmdNonStd, IORequest, DevicePtr;

VAR
  (* This variable must be initialized to point to the timer device before
   * the procedures in this module can be used!
   *)
  TimerBase : DevicePtr;

CONST
  (* unit definitions *)
  UnitMicroHz = 0;
  UnitVBlank   = 1;

  TimerName = "timer.device";

TYPE
  timevalPtr = POINTER TO timevalPtr;
  timeval = RECORD
    tvsecs : LONGCARD;
    tvmicro : LONGCARD;
  END;

  timerequestPtr = POINTER TO timerequest;
  timerequest = RECORD
    trnode : IORequest;
    trtime : timeval;
  END;

CONST
  (* IO_COMMAND to use for adding a timer *)
  TRAddRequest = CmdNonStd;
  TRGetSysTime = CmdNonStd + 1;
  TRSetSysTime = CmdNonStd + 2;

PROCEDURE AddTime(VAR Dest, Source: timeval);
(* Add one time request to another.

   Dest - a timeval structure
   Source - a timeval structure *)

PROCEDURE CmpTime(VAR Dest, Source: timeval): INTEGER;
(* Compare two timeval structures.

   Dest - a timeval structure
   Source - a timeval structure

   result - 0, if Dest has the same time as Source
            -1, if Dest has less time than Source
            +1, if Dest has more time than Source *)

PROCEDURE SubTime(VAR Dest, Source: timeval);
(* Subtract one time request from another.

   Dest - a timeval structure
   Source - a timeval structure

END TimerDevice.
```



```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: TrackDiskDevice.DEF                      Version: Amiga.00.00      *
* Created: 11/27/86   Updated: 12/31/86   Author: Leon Frenkel            *
* Description: "trackdisk.device" types.                                         *
*****)

```

```

DEFINITION MODULE TrackDiskDevice;

```

```

FROM SYSTEM IMPORT BYTE;
FROM IODevices IMPORT CmdNonStd, CmdClear, CmdRead, CmdUpdate, CmdWrite,
  IOStdReq, Unit;

```

```

CONST

```

```

  TDName = "trackdisk.device";

```

```

  (* Physical drive constants *)

```

```

  NumSecs = 11;

```

```

  NumUnits = 4;

```

```

  (* Usefull constants *)

```

```

  TDSector = 512;

```

```

  TDSecShift = 9; (* log TDSector *)

```

```

  TDMotor      = CmdNonStd + 0; (* control the disk's motor *)
  TDSeek       = CmdNonStd + 1; (* explicit seek (for testing) *)
  TDFormat     = CmdNonStd + 2; (* format disk *)
  TDRemove     = CmdNonStd + 3; (* notify when disk changes *)
  TDChangeNum  = CmdNonStd + 4; (* number of disk changes *)
  TDChangeState = CmdNonStd + 5; (* is there a disk in the drive? *)
  TDProtStatus = CmdNonStd + 6; (* is the disk write protected? *)
  TDRawRead    = CmdNonStd + 7; (* read raw bits to the disk *)
  TDRawWrite   = CmdNonStd + 8; (* write raw bits to the disk *)
  TDGetDriveType = CmdNonStd + 9; (* get the type of the disk drive *)
  TDGetNumTracks = CmdNonStd + 10; (* # of tracks for this type drive *)
  TDAddChangeInt = CmdNonStd + 11; (* TDRemove done right *)
  TDRemChangeInt = CmdNonStd + 12; (* remove softint set by TDAddChangeInt *)

```

```

  TDLastComm   = CmdNonStd + 13;

```

```

  TDExtCom     = 8000H; (* for internal use only! *)

```

```

  (* the disk driver has an "extended command" facility. These commands
   * take a superset of the normal IO Request block.
   *)

```

```

  ETDWrite     = TDExtCom + CmdWrite;
  ETDRead      = TDExtCom + CmdRead;
  ETDMotor     = TDExtCom + TDMotor;
  ETDSeek      = TDExtCom + TDSeek;
  ETDFormat    = TDExtCom + TDFormat;
  ETDUpdate    = TDExtCom + CmdUpdate;
  ETDClear     = TDExtCom + CmdClear;
  ETDRawRead   = TDExtCom + TDRawRead;
  ETDRawWrite  = TDExtCom + TDRawWrite;

```

```

TYPE

```

```

  (* extended IO has a larger than normal io request block. *)

```

```

  IOExtTDPtr = POINTER TO IOExtTD;

```

```

  IOExtTD = RECORD

```

```

    iotdReq      : IOStdReq;

```

```

    iotdCount    : LONGCARD;

```

```

    iotdSecLabel : LONGCARD;

```

```

  END;

```

Module TrackDiskDevice

CONST

```
(* raw read and write can be synced with the index pulse. This flag *)
(* in io request's ioFlags field tells the driver that you want this. *)
IOTDIndexSync = 4;
```

```
(* labels are TDLABELSIZE bytes per sector *)
TDLABELSIZE = 16;
```

```
(* This is a bit in the FLAGS field of OpenDevice. If it is set, then *)
(* the driver will allow you to open all the disks that the trackdisk *)
(* driver understands. Otherwise only 3.5" disks will succeed. *)
TDAllowNon35 = 0;
```

```
(* If you set the TDB_ALLOW_NON_3_5 bit in OpenDevice, then you don't *)
(* know what type of disk you really got. These defines are for the *)
(* TD_GETDRIVETYPE command. In addition, you can find out how many *)
(* tracks are supported via the TD_GETNUMTRACKS command. *)
Drive35 = 1; (* 3.5" *)
Drive525 = 2; (* 5.25" *)
```

```
(* Driver error defines *)
TDErrNotSpecified = 20; (* general catchall *)
TDErrNoSecHdr = 21; (* couldn't even find a sector *)
TDErrBadSecPreamble = 22; (* sector looked wrong *)
TDErrBadSecID = 23; (* ditto *)
TDErrBadHdrSum = 24; (* header had incorrect checksum *)
TDErrBadSecSum = 25; (* data had incorrect checksum *)
TDErrTooFewSecs = 26; (* couldn't find enough sectors *)
TDErrBadSecHdr = 27; (* another "sector looked wrong" *)
TDErrWriteProt = 28; (* can't write to a protected disk *)
TDErrDiskChanged = 29; (* no disk in the drive *)
TDErrSeekError = 30; (* couldn't find track 0 *)
TDErrNoMem = 31; (* ran out of memory *)
TDErrBadUnitNum = 32; (* asked for a unit > NumUnits *)
TDErrBadDriveType = 33; (* not a drive that trackdisk works *)
TDErrDriveInUse = 34; (* someone else allocated the drive *)
TDErrPostReset = 35; (* user hit reset; awaiting doom *)
```

TYPE

```
(* public portion of the unit structure *)
TDUPublicUnitPtr = POINTER TO TDUPublicUnit;
TDUPublicUnit = RECORD
    tduUnit : Unit; (* base message port *)
    tduComp01Track : CARDINAL; (* track for first precomp *)
    tduComp10Track : CARDINAL; (* track for second precomp *)
    tduComp11Track : CARDINAL; (* track for third precomp *)
    tduStepDelay : LONGCARD; (* time to wait after stepping *)
    tduSettleDelay : LONGCARD; (* time to wait after seeking *)
    tduRetryCnt : BYTE; (* #of times to retry *)
END;
```

END TrackDiskDevice.

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Translator.DEF                      Version: Amiga.00.00      *
* Created: 11/29/86   Updated: 12/31/86   Author: Leon Frenkel      *
* Description: Amiga "translator.library" definitions.                *
*****)

```

```

DEFINITION MODULE Translator;

```

```

FROM SYSTEM IMPORT ADDRESS;
FROM Libraries IMPORT LibraryPtr;

```

```

CONST
  TranslatorName = "translator.library";

```

```

VAR
  TranslatorBase : LibraryPtr;

```

```

CONST
  (* Translator error return codes *)
  TRNotUsed = -1; (* This is an oft used system rc *)
  TRNoMem    = -2; (* Can't allocate memory *)
  TRMakeBad  = -4; (* Error in MakeLibrary call *)

```

```

PROCEDURE Translate(instring: ADDRESS; inlen: LONGINT;
                   outbuf: ADDRESS; outlen: LONGINT): LONGINT;
(* Convert an English string into phonemes.

```

```

  instring - pointer to English string
  inlen    - length of English string
  outbuf   - a char array which will hold the phonetic codes
  outlen   - the length of the output array

```

```

  result - Translate will return a zero if no error has occurred.
  The only error that can occur is overflowing the output
  buffer. If Translate determines that an overflow will
  occur, it will stop the translation at a word boundary
  before the overflow happens. If this occurs, Translate
  will return a negative number whose absolute value
  indicates where in the INPUT string Translate stopped.
  The user can then use the offset -rtnCode from the
  beginning of the buffer in a subsequent Translate call
  to continue the translation where s/he left off. *)

```

```

END Translator.

```

Module Views

```
(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
*****
* Name: Views.DEF                               Version: Amiga.00.00
* Created: 11/20/86   Updated: 01/16/87   Author: Leon Frenkel
* Description: Graphics View types/functions.
*****)
```

DEFINITION MODULE Views;

```
FROM SYSTEM IMPORT ADDRESS, BYTE;
FROM System IMPORT GfxBase; (* Used by IMPLEMENTATION MODULE *)
FROM Copper IMPORT CopListPtr, cprlistPtr, UCopListPtr;
FROM Rasters IMPORT RasInfoPtr;
```

TYPE

```
ColorTablePtr = POINTER TO ColorTable;
ColorTable = ARRAY [0..31] OF CARDINAL;
```

```
ColorMapPtr = POINTER TO ColorMap;
ColorMap = RECORD
    Flags      : BYTE;
    Type       : BYTE;
    Count      : CARDINAL;
    ColorTable : ColorTablePtr;
END;
```

(* if Type = zero then ColorTable is a table of CARDINAL xRGB *)

```
ViewModes = (VMO,
    GenLockVideo, (* Gen Lock Video *)
    Lace,         (* Interlace 400 vertical resolution *)
    VM3,
    VM4,
    VM5,
    PFBA,         (* playfield priority *)
    ExtraHalfBrite, (* future use *)
    GenLockAudio, (* Gen Lock Audio *)
    VM9,
    DualPF,       (* Dual Play Field *)
    HAM,          (* (H)old (A)nd (M)odify *)
    VM12,
    VPHide,       (* reuse another plane ctr bit *)
    Sprites,      (* reuse one of plane ctr bits *)
    Hires);       (* Hires 640 width *)
```

ViewModesSet = SET OF ViewModes;

ViewPortPtr = POINTER TO ViewPort;

```
ViewPort = RECORD
    Next      : ViewPortPtr;
    ColorMap  : ColorMapPtr; (* table of colors for this viewport *)
                                (* if NIL MakeVPort assumes defaults *)
    DspIns    : CopListPtr; (* used by MakeView() *)
    SprIns    : CopListPtr; (* used by sprite stuff *)
    ClrIns    : CopListPtr; (* used by sprite stuff *)
    UCopIns   : UCopListPtr; (* user copper list *)
    DWidth    : INTEGER;
    DHeight   : INTEGER;
    DxOffset  : INTEGER;
    DyOffset  : INTEGER;
    Modes     : ViewModesSet;
    reserved  : CARDINAL;
    RasInfo   : RasInfoPtr;
END;
```

ViewPtr = POINTER TO View;

```
View = RECORD
    ViewPort : ViewPortPtr;
    LOFCprList : cprlistPtr; (* used for interlaced and noninterlaced *)
    SHFCprList : cprlistPtr; (* only used during interlace *)
    DyOffset   : INTEGER; (* for complete View positioning *)
    DxOffset   : INTEGER; (* offset are +- adjustments to stand #s *)
    Modes      : ViewModesSet; (* viden modes *)
END;
```

```

PROCEDURE FreeColorMap(VAR colormap: ColorMap);
(* Free the ColorMap structure and return memory to free memory pool.

    colormap - ColorMap allocated with GetColorMap *)

PROCEDURE FreeVPortCopLists(VAR vp: ViewPort);
(* Deallocate all intermediate copper lists and their headers from a viewport.

    vp - ViewPort structure *)

PROCEDURE GetColorMap(entries: LONGINT): ColorMapPtr;
(* Allocate and initialize Colormap.

    entries - number of entries for this colormap

    result - Pointer to a colormap or NIL if colormap could not be allocated *)

PROCEDURE GetRGB4(VAR colormap: ColorMap; entry: LONGINT): LONGCARD;
(* Inquire value of entry in ColorMap.

    colormap - ColorMap structure
    entry - index into colormap

    result - RGB value 4 bits per gun right justified or -1 if no valid entry *)

PROCEDURE InitView(VAR view: View);
(* Initialize view structure.

    view - a View structure *)

PROCEDURE InitVPort(VAR vp: ViewPort);
(* Initialize ViewPort structure.

    vp - a ViewPort structure *)

PROCEDURE LoadRGB4(VAR vp: ViewPort; colors: ADDRESS; count: INTEGER);
(* Load RGB color values from table.

    vp - a ViewPort, whos colors you want to change
    colors - pointer to of RGB values set up as an array of CARDINAL
    count - number of entries in table *)

PROCEDURE LoadView(VAR view: View);
(* Use a (possibly freshly created) coprocessor instruction list to create
    the current display.

    view - the View structure which contains the pointer to the
        constructed coprocessor instruction list *)

PROCEDURE MakeVPort(VAR view: View; VAR viewport: ViewPort);
(* Generate display copper list.

    view - View structure
    viewport - ViewPort structure *)

PROCEDURE MrgCop(VAR view: View);
(* Merge together coprocessor instructions.

    view - a view structure *)

PROCEDURE ScrollVPort(VAR vp: ViewPort);
(* Reinterpret RasInfo information in ViewPort.

    vp - a ViewPort structure that is currently displayed *)

```

Module Views

```
PROCEDURE SetRGB4(VAR vp: ViewPort; n: INTEGER; r, g, b: CARDINAL);
(* Set one color register for this viewport.
```

```
    vp - viewport structure
    n - the color number (range from 0 to 31)
    r - red level
    g - green level
    b - blue level *)
```

```
PROCEDURE SetRGB4CM(VAR cm: ColorMap; n: INTEGER; r, g, b: CARDINAL);
(* Set one color register for this ColorMap.
```

```
    cm - colormap
    n - the color number (range from 0 to 31)
    r - red level
    g - green level
    b - blue level *)
```

```
PROCEDURE WaitBOVP(VAR vp: ViewPort);
(* Wait till vertical beam reached bottom of this viewport.
```

```
    vp - ViewPort structure *)
```

```
END Views.
```

```

(*****
*                               *
*      Modula-2 Software Construction Set      *
*      (c) 1986 by Leon Frenkel                *
*                               *
* Name: Workbench.DEF                      Version: Amiga.00.00 *
* Created: 11/29/86   Updated: 05/07/87   Author: Leon Frenkel *
* Description: Amiga Workbench and "icon.library" definitions. *
*****)

```

```

DEFINITION MODULE Workbench;

```

```

FROM SYSTEM IMPORT ADDRESS, TSIZE;
FROM Intuition IMPORT Gadget, NewWindow;
FROM Libraries IMPORT LibraryPtr;
FROM Lists IMPORT List;
FROM Ports IMPORT Message, MsgPortPtr;

```

```

CONST

```

```

  IconName = "icon.library";

```

```

VAR

```

```

  IconBase : LibraryPtr;

```

```

TYPE

```

```

  DrawerDataPtr = POINTER TO DrawerData;
  DrawerData = RECORD
    ddNewWindow : NewWindow; (* args to open window *)
    ddCurrentX : LONGINT; (* current x coordinate of origin *)
    ddCurrentY : LONGINT; (* current y coordinate of origin *)
  END;

```

```

CONST

```

```

  (* the amount of DrawerData actually written to disk *)
  DrawerDataFileSize = TSIZE(DrawerData);

```

```

TYPE

```

```

  DiskObjectType = (WBO, (* not used *)
    WBDisk,
    WBDrawer,
    WBTool,
    WBProject,
    WBGarbage,
    WBDevice,
    WBKick);

```

```

  DiskObjectPtr = POINTER TO DiskObject;

```

```

  DiskObject = RECORD
    doMagic : CARDINAL; (* magic number at start of file *)
    doVersion : CARDINAL; (* version number, so it can change *)
    doGadget : Gadget; (* a copy of in core gadget *)
    doType : DiskObjectType;
    doDefaultTool : ADDRESS;
    doToolTypes : ADDRESS;
    doCurrentX : LONGINT;
    doCurrentY : LONGINT;
    doDrawerData : DrawerDataPtr;
    doToolWindow : ADDRESS; (* only applies to tools *)
    doStackSize : LONGINT; (* only applies to tools *)
  END;

```

```

CONST

```

```

  WBDiskMagic = OE310H; (* a magic number, not easily impersonated *)
  WBDiskVersion = 1; (* our current version number *)

```

```

TYPE

```

```

  FreeListPtr = POINTER TO FreeList;
  FreeList = RECORD
    f1NumFree : INTEGER;
    f1MemList : List;
  END;

```


Module Workbench

CONST

```
(* each message that comes into the WorkBenchPort must have a type file *)
(* in the preceding short. These are the defines for this type *)
MTypePStd      = 1; (* a "standard Potition" message *)
MTypeToolExit  = 2; (* exit message from out tools *)
MTypeDiskChange = 3; (* dos telling us of a disk change *)
MTypeTimer     = 4; (* we got a timer tick *)
MTypeCloseDown = 5; (* <unimplemented> *)
MTypeIOProc    = 6; (* <unimplemented> *)
```

```
(* workbench does different complement modes for its gadgets.
* It supports separate images, complement mode, and backfill mode.
* The first two are identical to intuitions GADGIMAGE and GADGHCOMP.
* backfill is similar to GADGHCOMP, but the region outside of the
* image (which normally would be color three when complemented)
* is flood-filled to color zero.
*)
```

```
GadgBackFill = 1;
```

```
(* if an icon does not really live anywhere, set it current position to here*)
NoIconPosition = 80000000H;
```

TYPE

```
WBArgPtr = POINTER TO WBArg;
WBArg = RECORD
    waLock : ADDRESS; (* a lock descriptor BPTR *)
    waName : ADDRESS; (* a string relative to that lock *)
END;
```

```
WBStartupPtr = POINTER TO WBStartup;
WBStartup = RECORD
```

```
    smMessage : Message; (* a standard message structure *)
    smProcess : MsgPortPtr; (* the process descriptor for you *)
    smSegment : ADDRESS; (* a descriptor for your code BPTR *)
    smNumArgs : LONGINT; (* number of elements in ArgList *)
    smToolWindow : ADDRESS; (* description of window *)
    smArgList : WBArgPtr; (* the arguments themselves *)
END;
```

```
PROCEDURE AddFreeList(VAR free: FreeList; mem: ADDRESS; len: LONGCARD): CARDINAL;
(* Add memory to the free list.
```

```
    free - a FreeList structure
    mem - the base of the memory to be recorded
    len - the length of the memory to be recorded

    result - nonzero if the call succeeded *)
```

```
PROCEDURE BumpRevision(newbuf: ADDRESS; oldname: ADDRESS): ADDRESS;
(* Reformat a name for a second copy.
```

```
    newbuf - the new buffer that will receive the name (minimum of 31 chars long)
    oldname - the original name

    result - a pointer to newbuf *)
```

```
PROCEDURE FindToolType(toolTypeArray: ADDRESS; typeName: ADDRESS): ADDRESS;
(* Find the value of a ToolType variable.
```

```
    toolTypeArray - an array of strings
    typeName - the name of the tooltype entry

    result - pointer to a string that is the value bound to typeName or NIL
             if typeName not in toolTypeArray *)
```

```
PROCEDURE FreeDiskObject(VAR diskobj: DiskObject);
(* Free all memory in a Workbench disk object.
```

```
    diskobj - a DiskObject structure *)
```



```
PROCEDURE FreeFreeList(VAR free: FreeList);
(* Free all memory in a free list.

   free - a FreeList structure *)

PROCEDURE GetDiskObject(name: ADDRESS): DiskObjectPtr;
(* Read in a Workbench disk object.

   name - name of the object

   result - the Workbench disk object in question *)

PROCEDURE GetIcon(name: ADDRESS; VAR icon: DiskObject;
                  VAR free: FreeList): CARDINAL;
(* Read in a DiskObject structure from disk.

   name - name of the object
   icon - a DiskObject structure
   free - a FreeList structure

   result - non-zero if the call succeeded *)

PROCEDURE MatchToolValue(typeString: ADDRESS; value: ADDRESS): CARDINAL;
(* Check a tool type variable for a particular value.

   typeString - a ToolType value (as returned by FindToolType)
   value - you are interested if value appears in typeString

   result - a one if the value was in typeString *)

PROCEDURE PutDiskObject(name: ADDRESS; VAR diskobj: DiskObject): CARDINAL;
(* Write out a DiskObject to disk.

   name - name of the object
   diskobj - a DiskObject structure

   result - non-zero if the call succeeded *)

PROCEDURE PutIcon(name: ADDRESS; VAR icon: DiskObject): CARDINAL;
(* Write out a DiskObject to disk.

   name - name of the object
   icon - a DiskObject structure

   result - non-zero if the call succeeded *)

END Workbench.
```

Appendix A Error Messages.....	3
Modula-2 Compiler Error Messages.....	3
List of Compiler Errors.....	3
Description of Compiler Errors.....	7
Description of Non Compilation Errors.....	21
Description of Fatal Errors.....	22
Modula-2 Linker Error Messages.....	23
EMACS Editor Error Messages.....	25
General Errors.....	25
Modula-2 Compiler Errors.....	28
Modula-2 Linker Errors.....	30
Modula-2 Run Errors.....	32
Miscellaneous Errors.....	33
Modula-2 Error Lister Error Messages.....	34
Modula-2 Object Module Generator.....	35

Appendix A Error Messages..... 3
Modula-2 Compiler Error Messages..... 3
List of Compiler Errors..... 3
Description of Compiler Errors..... 3
Description of Non-Compilation Errors..... 3
Description of Fatal Errors..... 3
Modula-2 Linker Error Messages..... 3
EMACS Editor Error Messages..... 3
General Errors..... 3
Modula-2 Compiler Errors..... 3
Modula-2 Linker Errors..... 3
Modula-2 Run Errors..... 3
Miscellaneous Errors..... 3
Modula-2 Error List Error Messages..... 3
Modula-2 Object Module Generator..... 3

This appendix contains a summary of the error messages generated by each of the software tools. For each error there is a detailed explanation of the possible reason for the error and in some cases an example.

Errors which relate to files are generally rather vague. This is because there are so many possible reasons why doing an operation on a given file may return an error. Typical file related errors are not specifying the correct file name, not having the correct disk in the drive, and many others. When encountering difficulties with file related errors refer to an AmigaDOS reference manual.

Modula-2 Compiler Error Messages

This subsection deals with the errors generated by the compiler. A compiler error consists of two component the error code and the error message. The error code is useful for quickly finding the description associated with a given error. The error message is a short message which identifies the problem. Many of the compiler errors are self explanatory. However, a more detailed description is provided help for each error.

Compilation errors can be displayed using the error lister program or from within the EMACS editor. Refer to the chapter on the error lister or the EMACS editor for additional information.

When trying to understand the reason for a given error closely examine the context in which the error occurred. Examining the context is necessary because many of the error messages are generic and may occur in many different situations.

Fatal errors and errors not directly related to compilation are displayed by the compiler directly to the console. These errors are listed in a separate subsection of this appendix.

List of Compiler Errors

This subsection lists the errors generated by the compiler. The errors are listed by the error code associated with the error. For more information refer to another subsection of this appendix which has a detailed description associated with each message.

Code Message

10	identifier expected
11	, comma expected
12	; semicolon expected
13	: colon expected
14	. period expected

15	)	right parenthesis expected
16	]	right bracket expected
17	}	right brace expected
18	=	equal sign expected
19	:=	assignment expected
20	END	expected
21	..	ellipsis expected
22	(	left parenthesis expected
23	OF	expected
24	TO	expected
25	DO	expected
26	UNTIL	expected
27	THEN	expected
28	MODULE	expected
29		invalid literal constant
30	IMPORT	expected
31		factor starts with illegal symbol
32		identifier, (, or [expected
33		identifier, ARRAY, RECORD, SET, POINTER, PROCEDURE, (, or [expected
34		Type followed by illegal symbol
35		statement starts with illegal symbol
36		declaration followed by illegal symbol
37		statement part is not allowed in definition module
38		export list not allowed in program module
39		EXIT not inside a LOOP statement
40		illegal character in number
41		number too large
42		comment without closing *)
43		unknown compiler switch
44		constant expression expected
45		string terminator not found on line
46		+ or - expected after compiler switch
50		identifier not declared or not visible
51		object should be a constant
52		object should be a type
53		object should be a variable
54		object should be a procedure
55		object should be a module
56		type should be a subrange
57		type should be a record
58		type should be an array
59		type should be a set
60		illegal base type of set
61		incompatible type of label or of subrange bound
62		multiply defined case (label)
63		low bound > high bound
64		more actual than formal parameters
65		fewer actual than formal parameters
66		more parameters in IMP than in DEF

67 too many constants in enumeration type
 68 mismatch between VAR specifications
 69 mismatch between type specifications
 70 more parameters in DEF than in IMP
 71 mismatch between result type specifications
 72 function in DEF, pure procedure in IMP
 73 procedure in DEF has parameters, but not in IMP
 74 code procedure cannot be defined FORWARD or defined in DEF module
 75 illegal type of control variable in FOR statement
 76 procedure call of a function
 77 identifiers in heading and at end do not match
 78 procedure already defined FORWARD
 79 body of FORWARD procedure undefined
 80 unsatisfied export list entry
 81 illegal type of procedure result
 82 illegal base type of subrange
 83 illegal type of case expression
 85 keys of imported symbol files do not match
 86 error in format of symbol file
 87 unresolved pointer type in current block
 88 symbol file not successfully opened
 89 procedure declared in definition module, but not in implementation
 90 illegal implementation of opaque type
 92 too many cases
 93 too many exit statements
 94 index type of array must be a subrange
 95 subrange bounds must be less than 2^{15}
 96 too many global modules
 98 too many structure elements in definition module
 100 multiple definition within the same scope
 107 illegal use of module
 108 constant index out of range
 109 indexed variable is not an array, or the index has the wrong type
 110 record selector is not a field identifier
 111 dereferenced variable is not a pointer
 112 operand type incompatible with sign inversion
 113 operand type incompatible with NOT
 114 $x \text{ IN } y$: type(x) # basetype(y)
 115 $x \text{ IN } y$: type of x cannot be the basetype of a set, or y is not a set
 116 {a..b}: type of either a or b is not equal to the base type of the set
 117 incompatible operand types
 118 operand type incompatible with *
 119 operand type incompatible with /
 120 operand type incompatible with DIV
 121 operand type incompatible with MOD
 122 operand type incompatible with AND
 123 operand type incompatible with +
 124 operand type incompatible with -
 125 operand type incompatible with OR

126	operand type incompatible with relation	67
127	procedure must have level 0	68
128	result type of P does not match that of T	69
129	mismatch of a parameter of P with the formal type list of T	70
130	procedure has fewer parameters than the formal type list	71
131	procedure has more parameters than the formal type list	72
132	assignment of a negative integer to a cardinal variable	73
133	incompatible assignment	74
134	assignment to non-variable	75
135	type of expression in IF, WHILE, UNTIL clause must be BOOLEAN	76
136	call of an object which is not a procedure	77
137	type of VAR parameter is not identical to that of actual parameter	78
138	constant outside of subrange bounds	79
139	type of RETURN expression differs from procedure type	80
141	step in FOR clause cannot be 0	81
142	illegal type of control variable	82
143	illegal type of FOR loop step	83
144	incorrect type of parameter of standard procedure	84
145	this parameter should be a type identifier	85
146	string is too long	86
147	incorrect priority specification	87
200	(not yet implemented)	88
201	integer too small for sign inversion	89
202	element outside of set range	90
203	overflow in multiplication	91
204	overflow in division	92
205	division by zero, or modulus with negative value	93
206	overflow in addition	94
207	overflow in subtraction	95
208	cardinal value assigned to integer variable too large	96
209	set size too large	97
210	array size too large	98
211	variable size too large	99
215	expression too complex (register overflow)	100
221	adr reloc buffer overflow	101
225	too many strings	102
226	program too long	103
227	identifier buffer overflow	104
228	heap overflow	105
230	expression not loadable	106
231	expression not addressable	107
232	expression not allowed	108
233	illegal expression	109
234	register reservation error	110
235	illegal selector for constant index field	111
236	too many WITH statements nested (> 4)	112
238	illegal size of operand	113
		114
		115
		116
		117
		118
		119
		120
		121
		122
		123
		124
		125

Description of Compiler Errors

This subsection contains detailed explanations of errors generated by the compiler.

Code Message

-
- 10 **identifier expected**
The compiler expected, but could not find an identifier.
- 11 **, comma expected**
The compiler expected, but could not find a comma.
- 12 **; semicolon expected**
The compiler expected, but could not find a semicolon.
- 13 **: colon expected**
The compiler expected, but could not find a colon.
- 14 **. period expected**
The compiler expected, but could not find a period.
- 15 **) right parenthesis expected**
The compiler expected, but could not find a right parenthesis.
- 16 **] right bracket expected**
The compiler expected, but could not find a right bracket.
- 17 **} right brace expected**
The compiler expected, but could not find a right brace.
- 18 **= equal sign expected**
The compiler expected, but could not find an equal sign.
- 19 **:= assignment expected**
The compiler expected, but could not find a colon followed by an equal sign.
- 20 **END expected**
The compiler expected, but could not find an END.
- 21 **.. ellipsis expected**
The compiler expected, but could not find a double period.
- 22 **(left parenthesis expected**
The compiler expected, but could not find a left parenthesis.
- 23 **OF expected**
The compiler expected, but could not find an OF.

- 24 **TO expected**
The compiler expected, but could not find a TO.
- 25 **DO expected**
The compiler expected, but could not find a DO.
- 26 **UNTIL expected**
The compiler expected, but could not find an UNTIL.
- 27 **THEN expected**
The compiler expected, but could not find a THEN.
- 28 **MODULE expected**
The compiler expected, but could not find a MODULE.
- 29 **invalid literal constant**
The specified literal constant is invalid. The characters which are valid in a literal constant depend on the type of constant being defined.
Example:
100 - long decimal constant
1.0 - floating point constant
0FC00H - hexadecimal constant
- 30 **IMPORT expected**
The compiler expected, but could not find an IMPORT.
- 31 **factor starts with illegal symbol**
An illegal symbol was found where an expression should begin.
- 32 **identifier, (, or [expected**
The compiler expected, but could not find a simple type which may be a type identifier, an enumeration or a subrange. A simple types begin with an identifier, a left parenthesis or a right bracket respectively.
- 33 **identifier, ARRAY, RECORD, SET, POINTER, PROCEDURE, (, or [expected**
The compiler expected, but could not find a type specification.
- 34 **Type followed by illegal symbol**
The compiler expected, but could not find a semicolon, a vertical bar, or an END following a type specification.
- 35 **statement starts with illegal symbol**
The compiler expected, but could not find a statement.
- 36 **declaration followed by illegal symbol**
The compiler expected, but could not find a symbol such as TYPE, VAR, PROCEDURE, BEGIN, END, etc...

- 37 **statement part is not allowed in definition module**
In the definition part of the module only the procedure heading are defined, the statements associated with a procedure are defined in the implementation part of the module.
- 38 **export list not allowed in program module**
An export list may only appear inside a local module.
- 39 **EXIT not inside a LOOP statement**
An EXIT statement is only valid inside a LOOP statement.
- 40 **illegal character in number**
An illegal character was encountered in the number specified.
- 41 **number too large**
The specified number is outside the legal range for this type of number.
- 42 **comment without closing *)**
Each open comment "(*" must have a corresponding close comment "*)". Comments may be nested to any level.
- 43 **unknown compiler switch**
An undefined compiler switch has been encountered. A compiler switch is specified inside a comment as follows: "(*\$x *)" where x is a letter representing the compiler switch to be effected.
- 44 **constant expression expected**
The compiler expected an expression with a constant value. A constant expression is any expression which does not require the compiler to generate executable code to calculate.
Example:
3 + 5
{1,5}
- 45 **string terminator not found on line**
A string literal must begin and end on the same line. To define a literal which is longer than one line use the "\" character at the end of the line. Any character with an ASCII value less than " " is considered to be a line terminator, so a control character or a TAB character imbedded in a string would cause this error to be generated. To place special characters into a string use the "\" commands, i.e. \t, \n.
- 46 **+ or - expected after compiler switch**
The compiler expects to find a plus or minus sign after the char specifying the compiler switch to be effected.
- 50 **identifier not declared or not visible**
The specified identifier is not known to the compiler. An identifiers must be declared or imported before being used. Standard Modula-2 identifiers do not require any declarations.

- 51 **object should be a constant**
The compiler is expecting an object of the class constant.
- 52 **object should be a type**
The compiler is expecting an object of the class type.
- 53 **object should be a variable**
The compiler is expecting an object of the class variable.
- 54 **object should be a procedure**
The compiler is expecting an object of the class procedure.
- 55 **object should be a module**
The compiler is expecting an object of the class module.
- 56 **type should be a subrange**
The compiler is expecting a subrange type.
- 57 **type should be a record**
The compiler is expecting a record type.
- 58 **type should be an array**
The compiler is expecting an array type.
- 59 **type should be a set**
The compiler is expecting a set type.
- 60 **illegal base type of set**
A set type must be based on a subrange or an enumeration.
Example:
SET OF [0..10] (subrange based)
SET OF (a,b,c) (enumeration based)
- 61 **incompatible type of label or of subrange bound**
The upper and lower bounds of a subrange or label range must be of the same type.
Example:
[0..10] (compatible)
[0.."A"] (incompatible)
- 62 **multiply defined case (label)**
Each label in a case statement or case variant record must be unique, this applies to individual labels and a range of labels.
Example:
"0..10,5" would cause "5" to be multiply defined because "5" is already specified in the range "0..10".

- 63 **low bound > high bound**
 The lower bounds of the subrange is higher than the upper bounds of the subrange. This error also applies to a label range in a CASE statement or a variant record.
 Example:
 [10..5] (invalid)
 [5..10] (valid)
- 64 **more actual than formal parameters**
 The number of parameters specified in the procedure call is more than the number of parameters defined by the procedure.
 Example:
 PROCEDURE Draw(x,y: CARDINAL);
 ...
 Draw(10,20,30); (* too many parameters *)
- 65 **fewer actual than formal parameters**
 Not all of the parameters defined by the procedure are specified in the procedure call.
 Example:
 PROCEDURE Move(x,y: CARDINAL);
 ...
 Move(a+b); (* not enough parameters *)
- 66 **more parameters in IMP than in DEF**
 The procedure in the implementation module has been specified with more parameters than are declared in the procedure heading in the definition part of the module.
 Example:
 Definition -> PROCEDURE sqrt(x: REAL);
 Implementation-> PROCEDURE sqrt(x: REAL; y: INTEGER);
- 67 **too many constants in enumeration type**
 An enumeration type is not allowed to have more than 256 constants.
- 68 **mismatch between VAR specifications**
 A procedure in the implementation module has been specified as a VAR parameter while the procedure in the definition module has been specified as a value parameter, or vice versa.
 Example:
 Definition -> PROCEDURE sqrt(x: REAL);
 Implementation -> PROCEDURE sqrt(VAR x: REAL);

- 69 **mismatch between type specifications**
A procedure parameter in the implementation module has been specified as being of a different type than a procedure parameter in the definition module.
Example:
Definition -> PROCEDURE sqrt(x: REAL);
Implementation -> PROCEDURE sqrt(x: LONGINT);
- 70 **more parameters in DEF than in IMP**
The procedure in the implementation module has been specified with fewer parameters than are declared in the procedure heading in the definition part of the module.
Example:
Definition -> PROCEDURE sqrt(x: REAL t: LONGCARD);
Implementation -> PROCEDURE sqrt(x: REAL);
- 71 **mismatch between result type specifications**
The procedure heading in the definition module specified a different return parameter type than the procedure defined in the implementation module.
Example:
Definition -> PROCEDURE sqrt(x: REAL): REAL;
Implementation -> PROCEDURE sqrt(x: REAL): LONGINT;
- 72 **function in DEF, pure procedure in IMP**
The procedure heading in the definition module specified a return parameter for a procedure, but the procedure defined in the implementation module does not specify a return parameter.
Example:
Definition -> PROCEDURE sqrt(x: REAL): REAL;
Implementation -> PROCEDURE sqrt(x: REAL);
- 73 **procedure in DEF has parameters, but not in IMP**
The procedure heading in the definition modules is specified as having parameters, but the procedure defined in the implementation part is specified as not having any parameters.
Example:
Definition -> PROCEDURE sin(x: REAL);
Implementation -> PROCEDURE sin;
- 74 **code procedure cannot be defined FORWARD or defined in DEF module**
There are two possible reasons for this error. An attempt is being made to defined a CODE procedure as FORWARD or the CODE procedure has already defined in the definition part of the library module currently being compiled.
- 75 **illegal type of control variable in FOR statement**
The control variable in a FOR statement can not be a component of a structured variable, it can not be imported, and it can not be a parameter.

- 76 **procedure call of a function**
A procedure which specifies a return value can not be invoked via a procedure call statement, it can only be called from an expression.
- 77 **identifiers in heading and at end do not match**
The identifier specified at the beginning of a procedure or module is different from the identifier at the end of the module. Both identifiers must be the same.
Example:
PROCEDURE Init;
BEGIN
...
END Initialize; (* should be "Init" *)
- 78 **procedure already defined FORWARD**
The procedure currently being defined as FORWARD has already been defined as FORWARD earlier in the source file.
- 79 **body of FORWARD procedure undefined**
A procedure has been defined as FORWARD, but no corresponding procedure body has been defined in the current scope. This error occurs when the compiler reaches the end of the current scope. To eliminate the error examine the current scope for FORWARD definitions without corresponding procedure body definitions at the same scope.
- 80 **unsatisfied export list entry**
An identifier specified in the export list of the current local module is not visible in the current local module.
- 81 **illegal type of procedure result**
The specified type conversion is illegal.
Example:
NodeRec(10), where NodeRec is a RECORD type, is illegal.
- 82 **illegal base type of subrange**
The specified type is not valid as the base type of a subrange.
- 83 **illegal type of case expression**
The case expression is of a type that is invalid in a case statement.
- 85 **keys of imported symbol files do not match**
One or more of the modules being imported have not been recompiled after a definition module they depended on was changed, this has caused a key conflict between the modules.
- 86 **error in format of symbol file**
The symbol file currently being loaded is not a valid symbol file.

- 87 **unresolved pointer type in current block**
A pointer type has been forward defined but has not been subsequently defined in the current block.
Example:
TYPE
 MyObjPtr = POINTER TO MyObj; (* Forward Pointer Def *)
- 88 **symbol file not successfully opened**
An attempt to open a symbol file failed. Most likely the file was not found in the search through the symbol file directory(s) or the operating system may have locked it from being read.
- 89 **procedure declared in definition module, but not in implementation**
A procedure declared in the definition module has not been declared in the implementation module.
- 90 **illegal implementation of opaque type**
In this as in most implementations of Modula-2 an opaque type is limited in the types which it can represent. An opaque type is usually defined as a pointer, but in this implementation any simple type which is exactly 4 bytes long is valid.
- 92 **too many cases**
The compiler allows a maximum of 128 cases or variants per case statement and variant record respectively.
- 93 **too many exit statements**
The compiler allows a maximum of 16 EXIT statements per LOOP..END block.
- 94 **index type of array must be a subrange**
The index specification for an array type must be a subrange.
- 95 **subrange bounds must be less than 2^{15}**
The bounds of a subrange type must be in the range of -32768 and 32767.
- 96 **too many global modules**
The module that is being imported is too complex and should be broken up into smaller modules.
- 98 **too many structure elements in definition module**
The module that is being imported is too complex and should be broken up into smaller modules.
- 100 **multiple definition within the same scope**
The identifier being declared already exists at the current scope. A new name should be chosen which would not conflict with the identifier already declared at the current scope level.

- 107 **illegal use of module**
No operations are possible on module identifiers.
- 108 **constant index out of range**
An array index has been specified which is outside the bounds of the array.
Example:
VAR LookupTable : ARRAY [0..100] OF REAL;
...
LookupTable[300] := 1.2; (300 is out of bounds of the array)
- 109 **indexed variable is not an array, or the index has the wrong type**
The variable that is being indexed is not of an array type or the index is not compatible with the type of the array index.
Example:
VAR list : POINTER TO Node;
...
list[0] := NodePtr; (* "list" is not an array! *)
- 110 **record selector is not a field identifier**
The specified identifier is not a field of the record being referenced.
Example:
VAR node: RECORD x,y: CARDINAL; END;
...
node.z := 10; (* "z" is not defined as a field *)
- 111 **dereferenced variable is not a pointer**
The variable being dereferenced is not a pointer, the dereference "^" operation can only be performed on a variable of a pointer type.
Example:
VAR ch : CHAR;
...
ch^ := "a"; (* "ch" is not a pointer *)
- 112 **operand type incompatible with sign inversion**
The negate operation only works for the following types: INTEGER, LONGINT or REAL. All other types are invalid.
- 113 **operand type incompatible with NOT**
The NOT operation only works for the BOOLEAN type. All other types are invalid.
- 114 **x IN y: type(x) # basetype(y)**
The IN operation requires that the element being tested be the same as the base type of the set.
Example:
x IN {4,6,10} is valid only if x is an INTEGER or CARDINAL.
- 115 **x IN y: type of x cannot be the basetype of a set, or y is not a set**
The expression "y" is not of a set type or "x" is not of a type which can be the basetype of a set.

- 116 **{a..b}: type of either a or b is not equal to the base type of the set**
Either the lower or upper bounds of the set range is not compatible with the base type of the set.
- 117 **incompatible operand types**
The specified operands are not compatible with each other.
- 118 **operand type incompatible with ***
One or both of the operands are incompatible with the "*" operator.
- 119 **operand type incompatible with /**
One or both of the operands are incompatible with the "/" operator. The "/" operator is only valid for REAL or SET operand types.
- 120 **operand type incompatible with DIV**
One or both of the operands are incompatible with the "DIV" operator.
- 121 **operand type incompatible with MOD**
One or both of the operands are incompatible with the "MOD" operator.
- 122 **operand type incompatible with AND**
One or both of the operands are incompatible with the "AND" operator. The "AND" operator is only compatible with BOOLEAN operand types.
- 123 **operand type incompatible with +**
One or both of the operands are incompatible with the "+" operator.
- 124 **operand type incompatible with -**
One or both of the operands are incompatible with the "-" operator.
- 125 **operand type incompatible with OR**
One or both of the operands are incompatible with the "OR" operator. The "OR" operator is only compatible with BOOLEAN operand types.
- 126 **operand type incompatible with relation**
One or both of the operands are incompatible with relational operator.
- 127 **procedure must have level 0**
Only a procedure which is not enclosed by any procedure can be assigned to a procedure variables or passed as a parameter.
- 128 **result type of P does not match that of T**
In order to assign the name of a procedure to a variable or pass a procedure as a parameter the parameter lists of the variable and the procedure must be identical.
- 129 **mismatch of a parameter of P with the formal type list of T**
In order to assign the name of a procedure to a variable or pass a procedure as a parameter the parameter lists of the variable and the procedure must be identical.

- 130 **procedure has fewer parameters than the formal type list**
In order to assign the name of a procedure to a variable or pass a procedure as a parameter the parameter lists of the variable and the procedure must be identical.
- 131 **procedure has more parameters than the formal type list**
In order to assign the name of a procedure to a variable or pass a procedure as a parameter the parameter lists of the variable and the procedure must be identical.
- 132 **assignment of a negative integer to a cardinal variable**
A negative integer constant can not be assigned to a cardinal variable. The above only applies to integer constants. Integer variables can be assigned to cardinal variables and vice-versa.
- 133 **incompatible assignment**
The object being assigned is not assignment compatible with the variable to which it is being assigned.
- 134 **assignment to non-variable**
The left side of an assignment statement must contain a variable identifier or designator.
- 135 **type of expression in IF, WHILE, UNTIL clause must be BOOLEAN**
The expressions for the statements IF, WHILE, UNTIL expect an expression of a BOOLEAN type.
- 136 **call of an object which is not a procedure**
The object being called is not a procedure.
- 137 **type of VAR parameter is not identical to that of actual parameter**
If a procedure specifies a parameter as VAR the actual parameter passed to a procedure must be identical to the type of the VAR parameter.
- 138 **constant outside of subrange bounds**
When assigning a constant to a variable of a subrange type the compiler performs range checking of the constant with the subrange of the variable.
- 139 **type of RETURN expression differs from procedure type**
The expression specified in a RETURN statement must be assignment compatible with the type of the return parameter.
- 141 **step in FOR clause cannot be 0**
When the optional BY clause is used in a FOR statement the constant expression which follows BY must not be zero.
- 142 **illegal type of control variable**
The control variable in a FOR loop must be one of the following: enumeration, BOOLEAN, CHAR, CARDINAL, INTEGER, LONGINT, or LONGCARD.

- 143 **illegal type of FOR loop step**
The BY clause of a FOR statement specifies the increment of the FOR loop. The constant expression must be one of the following types: INTEGER, CARDINAL, LONGINT, or LONGCARD.
- 144 **incorrect type of parameter of standard procedure**
The parameter passed to the standard procedure was not of the correct type.
- 145 **this parameter should be a type identifier**
The standard procedure called expects a type identifier as a parameter, not a numerical expression.
- 146 **string is too long**
The specified string literal is too long to be assigned to the fixed size array variable.
- 147 **incorrect priority specification**
The module priority specification should be a CARDINAL constant between 0 and 15.
- 200 **(not yet implemented)**
The source program uses an unimplemented function in the compiler. This error should not occur under normal circumstances.
- 201 **integer too small for sign inversion**
A constant integer with a value of -32768 can not be negated because the resulting value would be 32768 which is outside the range of integers.
- 202 **element outside of set range**
The specified element was outside the range defined by the set.
Example:
T = SET OF [10..30]; T{5} (outside of set range)
- 203 **overflow in multiplication**
Two constant operands were multiplied resulting in an overflow.
- 204 **overflow in division**
Two constant operands were divided resulting in an overflow.
- 205 **division by zero, or modulus with negative value**
Two constant operands were divided resulting in a division by zero or the right operand in a modulus operation has a negative value.
- 206 **overflow in addition**
Two constant operands were added resulting in an overflow.
- 207 **overflow in subtraction**
Two constant operands were subtracted resulting in an overflow.

- 208 **cardinal value assigned to integer variable too large**
A cardinal constant greater than 32767 can not be assigned to an integer as this is obviously an overflow of an integer.
- 209 **set size too large**
The specified set size is too large. The legal range for set elements is 0..31. The size of the set will be adjusted automatically based on the upper bounds of the range.
Example:
SET OF [0..7]; (1 byte)
SET OF [0..15] (2 bytes)
SET OF [0..31] (4 bytes)
- 210 **array size too large**
The specified array size is too large. The maximum size of an array is 32K bytes.
- 211 **variable size too large**
Too much space has been allocated for variables local to a procedure or module. Each module and each procedure can have up to 32K bytes of variables declared.
- 215 **expression too complex (register overflow)**
The specified expression is too complex for the compiler to handle. To fix the problem simply break up the expression into two or more separate expressions.
- 221 **adr reloc buffer overflow**
The compiler's address relocation buffer has overflowed. This can occur when your program references too many variables from other modules. To compile a program which encounters this problem, the compiler's command line option "-a <AdrBufferSize>" maybe used. The size of this buffer may be increased permanently using the compiler configuration utility.
- 225 **too many strings**
The compiler's string buffer has overflowed. This can occur when a program uses too many literal string constants. To compile a program which encounters this problem, the compiler's command line option "-b <StringBufferSize>" maybe used. The size of this buffer maybe increased permanently using the compiler configuration utility.
- 226 **program too long**
The compiler's code buffer has overflowed. This can happen when a very large program is compiled. To compile a large program the compiler's command line option "-c <CodeBufferSize>" maybe used. The size of this buffer maybe increased permanently using the compiler configuration utility. If the module is still too long then the module should be split up into two or more modules.

- 227 **identifier buffer overflow**
The compiler's identifier buffer has overflowed. This can happen when too many large symbol modules are imported or when a large program is compiled. To compile a large program the compiler's command line option "-i <IdentBufferSize>" maybe used. The size of this buffer maybe increased permanently using the compiler configuration utility.
- 228 **heap overflow**
The compiler's heap has overflowed. This can happen when too many large symbol modules are imported or when a large program is compiled. To compile a large program the compiler's command line option "-h <HeapSize>" maybe used. The HeapSize has no limit except physical memory in the system. The size of the heap maybe increased permanently using the compiler configuration utility.
- 230 **expression not loadable**
Internal compiler error occurred in a redundancy check.
- 231 **expression not addressable**
An attempt is being made to determine the address of an object which does not have an address. For example using the ADR() standard function on a constant identifier. However, in this implementation it is possible to take the address of a string constant.
- 232 **expression not allowed**
Internal compiler error occurred in a redundancy check.
- 233 **illegal expression**
Internal compiler error occurred in a redundancy check.
- 234 **register reservation error**
Internal compiler error occurred in a redundancy check.
- 235 **illegal selector for constant index field**
Internal compiler error occurred in a redundancy check.
- 236 **too many WITH statements nested (> 4)**
A maximum of four WITH statements can be nested at one time.
- 238 **illegal size of operand**
Internal compiler error occurred in a redundancy check.

Description of Non Compilation Errors

If the compiler is invoked from the CLI problems encountered by the compiler will be displayed on the console. These errors may not be directly related to the compilation. The following is a list of the possible error messages.

Errors in Source File!

This error indicates that the source file contained compilation errors. The error lister program should be run to display the actual errors encountered in the source file.

Error Opening Source File!

The specified source file can not be found. The probable cause of this error is that the wrong filename was specified for the name of the source file.

Error Opening or Reading Symbol File!

The symbol module file of a module being imported in the source file can not be found. This error usually occurs when the modules of a program are not compiled in the correct order. To fix the problem recompile the modules again in the proper order based on the module dependencies.

Error Opening Reference or Symbol File!

This error occurs when the compiler has difficulty opening an output file for a reference file or symbol file. This error is usually the result of attempting to output to a write protected disk or the disk has become full.

Error Writing Reference or Symbol File!

This error occurs when the compiler encounters a disk error while writing out a reference or symbol file. This error may be the result of a disk becoming full.

Error Opening or Writing Object File!

This error occurs when the compiler encounters difficulties writing out an object file. This error may be the result of a write protected disk or a disk becoming full.

Description of Fatal Errors

If the compiler is invoked from the CLI and a fatal error is encountered it will be displayed to the console. A fatal error is one which causes the compiler to abort and return to the CLI before it has completed its work. Each error message is preceded by "FATAL ERROR:" to alert the user of the seriousness of the error.

Not Enough Memory to Continue!

This error indicates that the compiler can not continue because it has run out of memory. A possible solution to this problem is to reduce the memory requirements of the compiler by using compiler command line options, such as, "-h" to reduce the compiler's heap size.

Invalid Command Line Argument:

This error indicates that a command line option specified on the command line was invalid. An option may be considered invalid for one of several reasons. One possible reason is that it exceeds the maximum value allowed for that option. The invalid option is listed after the fatal error message. Refer to the chapter on the compiler for details of each compiler option.

Opening Error Log File 'T:M2.err'!

This error indicates that the compiler was not able to open an error log file. This error may occur for one of several reasons. The assignment for "T:" may not have been setup. Another possible reason is that the compiler has already been invoked from another CLI, this means that the "M2.err" file is already in use.

Modula-2 Linker Error Messages

Most of the error messages produced by the linker are self explanatory. However, as further help this section contains a detailed explanation of each error message. Linker error messages begin with "ERROR:" to indicate that the information being displayed is an error and not just information which may be ignored. An error will cause the linker to abort the linking of the current program. However if more than one program is specified to be linked, the linker will begin linking the next program specified on the command line. To abort a batch link after getting an error use the Ctrl-C key combination which will cause the linker to stop upon finishing linking the current program. Refer to the chapter of this manual which describes the linker for additional information.

Can't Open Obj File '<FileName>'

An imported object module file could not be opened. The most probable reason for the error is the file was not found. The file may not have been found if the source file has not been compiled.

Obj File '<FileName>' is Invalid!

The specified object module file contains invalid information. This may occur if the object file being read is empty or contains garbage for some reason. If the file in question is one of the library modules included in this package restore the file from the original disk. Otherwise recompile the file and try another link. This error should be very rare.

Obj File '<FileName>' Wrong Version!

The specified object module file was compiled with a version of the compiler which is not compatible with this version of the linker. The solution is to recompile the original file with a newer version of the compiler.

Module '<ModuleName>' Has Bad Key!

The specified object module file is incompatible with the other modules being imported. Specifically, one of the previous modules has already imported an older version of this module. Refer to the chapter on the linker for additional information on the version control system and what to do when a bad key is encountered.

Not Enough Memory To Continue!

The linker has run out of memory while linking. The linker is very memory efficient, however this error may occur if other tasks are taking up a lot of memory. The solution to this problem is to free up some memory and try the link operation again.

Module '<ModuleName>' Imported By '<ModuleName>' Has Bad Key!

The first <ModuleName> is imported by the module specified by the second <ModuleName>. However, the file has already been imported by another module using a different key. Refer to the chapter on the linker for additional information on the version control system and what to do when a bad key is encountered.

Can't Open Output File '<FileName>'

An output file could not be created. This error may occur if the disk specified for output is write protected.

Module 'System' Not Imported!

After all of the modules were imported the module **System** was not one of the modules imported. This error should not occur under normal circumstances. However, it may occur when linking one assembly language module with no import of **System**.

Sym File '<FileName>' is Invalid!

The symbol or reference file read by the linker is not valid. A possible reason for this error is a corrupt file, empty file or a file generated by an incompatible version of the compiler.

Sym File '<FileName>' Has Bad key!

The symbol or reference file read by the linker has a key which is incompatible with the corresponding object module imported earlier. The solution to this problem is to recompile to get an up to date symbol or reference file.

EMACS Editor Error Messages

The EMACS editor displays error messages on the bottom line of the display, called the echo line. This section is divided into three subsection which describe the EMACS error messages as they relate to editing, compiling, linking and miscellaneous operations.

General Errors

This subsection contains a description of the error messages produced by EMACS that relate to editing, windows, files and buffers.

Display unusable

The EMACS window has been made too small for the number of buffers that are currently being displayed. Increase the size of the window and the display will work correctly again.

Cannot split a # line window

Windows that have less than three lines can not be split. This message indicates that you are trying to split a window with less than three display lines.

Can't get #

This message indicates that EMACS has run out of memory while trying to open a window. Free up memory by deleting some buffers.

Only one window

This message indicates that the window operation that is being attempted is inappropriate for a display with only one window open. For instance trying to increase the size of only one window is not possible.

Impossible change

This message indicates that the window operation that is being attempted is not possible. For example trying to shrink a window to less than one line.

Keyboard macro overflow

This message indicates that the current keyboard macro being defined is too long for EMACS to handle.

No mark set in this window

This error will occur when attempting a region operation before the "mark" has been set for that window. The solution is to specify a location for the "mark" and then perform the region operation.

Region is too large

The current region operation uses a region that is too large to be manipulated. The solution is to perform the operation in two or more steps using smaller regions.

[No match]

This message may appear when EMACS can not find the object referred to by a given name. This error applies to commands, buffers and other named objects.

eval-expression macro overflow

An overflow has occurred during the operation of the eval-expression function. Reduce the size of the expression and try the operation again.

Can't get # bytes

This message indicates that EMACS has run out of memory while trying to perform an operation. Free up memory by deleting some buffers.

Already defining kbd macro!

This error is displayed if an attempt is made to begin a new macro definition, while EMACS is in the macro definition mode.

Not defining kbd macro.

This error is displayed if an attempt to end a macro definition is made while not in the macro definition mode.

Not now

This error is displayed if an attempt is made to execute a macro, while EMACS is in the macro definition mode.

Search failed: "<String>"

This message is displayed if the previous search operation did not find the search string in the buffer.

No last search

This message is display if the search-again command is issued and a search string has not been previously defined.

Pattern too long

The string pattern to be searched for is too long. Specify a shorter search pattern.

No other window

The previous command was directed towards another window, while the display currently contains only one window.

Word too long!

A single word containing more than 256 characters was encountered while filing a paragraph. This error should not be encountered under normal circumstances.

Cannot insert buffer into self

The previous operation attempted to insert the current buffer into itself. This operation does not make sense and is illegal.

No file name

This message appears when a buffer save operation is performed on a buffer which does not have an associated file name. To save the buffer use the write-file command.

Can't find <FontName> <FontSize>!

The requested font of the specified size could not be found on disk. Examine the "fonts:" directory for the names of fonts available to the system.

Cannot open file for writing

The previous save operation failed because the output file could not be created. This error may occur due to a disk being write protected or the disk being full.

Write I/O error

The previous save operation failed because of an input/output error. This error may be the result of a disk becoming full or other disk related problems.

File has long line

The file that was read in the previous load operation contained one or more lines which contained too many characters for EMACS to handle. This type of error may occur when reading in a binary file or a file which does not have newline characters at the end of each line.

File read error

During the previous read operation an error was encountered. This may be the result of a corrupted file or other disk related problems.

Modula-2 Compiler Errors

This subsection contains a description of the error messages produced by EMACS which are related to the Modula-2 Compiler. Each error message is preceded with "ERROR:" to get the users attention.

Modula-2 Compiler Already Loaded!

The compiler has already been loaded and made resident. If the reason for reloading the compiler is to specify new command line parameters then the compiler should first be unloaded and then loaded again.

Insufficient Memory to Allocate Stack for the Modula-2 Compiler!

Not enough memory is available to allocate a stack for the Modula-2 compiler. The reason for this problem could be that many large text files have been loaded into EMACS. Check the amount of memory currently available by using the free-mem-avail command and taking appropriate actions to free up additional memory.

Modula-2 Compiler Not Found!

The Modula-2 compiler was not found in the current directory, in any of the directories specified with the AmigaDOS PATH command or the "c:" directory. Put a copy of the Modula-2 compiler into one of the above specified directories and try the command again.

Insufficient Memory to Load the Modula-2 Compiler!

The Modula-2 compiler could not be loaded because not enough free memory is currently available in the system. Free up additional memory by removing other tasks or erasing some EMACS buffers.

Modula-2 Compiler Loading Problem!

The Modula-2 compiler could not be loaded for some unknown reason. A possible reason would be if the current directory contained a file called "M2" which contained text instead of the compiler code. Another similar problem could be a directory called "M2" in either the current directory or one of the directories in the AmigaDOS PATH.

Invalid Arg String Passed to Modula-2 Compiler!

The argument string passed to the compiler was invalid. The compiler is unloaded. Refer to the chapter on the compiler for information on the options accepted by the compiler.

Insufficient Memory To Initialize Modula-2 Compiler!

The compiler ran out of memory while initializing. The compiler is automatically unloaded from memory. Free some additional memory and reload the compiler again.

Buffer Not Found!

The name of the buffer specified to be compiled could not be found. Examine the buffer list to find the exact name of the buffer that needs to be compiled.

Stack Size Can Not Be Changed Now!

EMACS does not allow the size of the compiler's stack to be altered while the compiler is loaded. To change the size of the compiler's stack unload the compiler, change the stack size and reload the compiler again.

Buffer '<BufferName>' Not Found! (Error List Discarded)

This message will appear when issuing a command to jump to an error position in a buffer which has been deleted from EMACS. The <BufferName> of the buffer containing the error position is displayed in the message. The error list becomes invalid when a buffer is deleted and is therefore discarded by EMACS.

File '<FileName>' Not Loaded In Buffer!

This message will appear when issuing a command to jump to an error position in a file which has not been loaded into a buffer.

File '<FileName>' Not Loaded In Buffer! (Error List Discarded)

This message will appear when issuing a command to jump to an error position in a file which has been loaded into a buffer, but the buffer has since been deleted. The error list becomes invalid when a buffer is deleted and is therefore discarded by EMACS.

Compilation Aborted. File '<FileName>' Not Found!

This message will be displayed if the compiler could not find the file specified to be compiled. Refer to the directory of the disk for the correct file name of the file that needs to be compiled.

Compilation Aborted. Insufficient Memory to Compile Program!

The compiler dynamically allocates memory for various tables before beginning compilation, this error will occur if there is not enough memory for the sizes of tables which need to be allocated. The solution to this problem is to reduce the sizes of the tables using the interactive command "set-m2-comp-opt" or by reloading the compiler with a new argument string. Also it may be possible to delete some unused EMACS buffers freeing up additional memory.

Compilation Aborted. Compiler's Error Log File Not Opened!

The compiler saves error to the error log file called "M2.err" in the directory "T:". A possible reason for this error is that the assignment for the "T:" directory has not been performed. Compilation can not take place unless this file can be opened successfully.

Opening or Writing Output File Failed!

The compilation was successful but the compiler encountered an error while trying to open or write the output file. This error may occur due to the disk being write protected or the disk may be full. Refer to the AmigaDOS reference manual for additional information.

Modula-2 Linker Errors

This subsection contains a description of the error messages produced by EMACS which are related to the Modula-2 Linker. Each error message is preceded with "ERROR:" to get the users attention.

The Main Module Has Not Been Specified!

This error indicates that the "main" module has not been specified. Use the command set-m2-main-module to specify the name of the "main" module to be used.

Unable To Open '<FileName>' As Output File!

The linker was not able to open the <FileName> as the output file. This may occur if the directory specified for output files does not exist or other disk related problems have occurred.

Insufficient Memory To Continue Linking Process!

The linker has run out of memory while linking a program. This should not happen very often because the linker is very memory efficient. However if the problem does occur free up some memory and try the link operation again.

Sym File '<FileName>' Has Bad Key!

The symbol or reference file read by the linker has a key which is incompatible with the corresponding object module. The solution to this problem is to recompile to get an up to date symbol or reference file.

Sym File '<FileName>' is Invalid!

The symbol or reference file read by the linker is not valid. A possible reason for this error is a corrupt file, empty file or a file generated by an incompatible version of the compiler.

Can't Open Obj File '<FileName>'!

An imported object module file could not be opened. The most probable reason for the error is the file was not found. The file may not have been found if the source file has not been compiled.

Obj File '<FileName>' is Invalid!

The specified object module file contains invalid information. This may occur if the object file being read is empty or contains garbage for some reason. If the file in question is one of the library modules included in this package restore the file from the original disk. Otherwise recompile the file and try another link. This error should be very rare.

Obj File '<FileName>' Wrong Version!

The specified object module file was compiled with a version of the compiler which is not compatible with this version of the linker. The solution is to recompile the original file with a newer version of the compiler.

Module '<ModuleName>' Has Bad Key!

The specified object module file is incompatible with the other modules being imported. Specifically, one of the previous modules has already imported an older version of this module. Refer to the chapter on the linker for additional information on the version control system and what to do when a bad key is encountered.

Module '<ModuleName>' Imported By '<ModuleName>' Has Bad Key!

The first <ModuleName> is imported by the module specified by the second <ModuleName>. However, the file has already been imported by another module using a different key. Refer to the chapter on the linker for additional information on the version control system and what to do when a bad key is encountered.

Module 'System' Not Imported!

After all of the modules were imported the module **System** was not one of the modules imported. This error should not occur under normal circumstances. However, it may occur when linking one assembly language module with no import of **System**.

Modula-2 Run Errors

This subsection contains a description of the error messages produced by EMACS which are related to running Modula-2 programs. Each error message is preceded with "ERROR:" to get the users attention.

Program '<FileName>' Not Found!

An attempt to run a user program failed because the specified <FileName> was not found. A possible reason for this error is that the program was never linked and so an executable file has not been generated. Another possible reason is that there is not enough memory to load the program into memory.

Can't Open Input/Output Window!

An attempt to open an input/output for the user program has failed. One possible reason for this problem is that there was insufficient memory to open the window. Another possible reason is that the window specification was incorrect. Check the current window specification using the "set-m2-run-window" command and change if necessary.

Example:

```
RAW:0/0/320/100/
```

```
CON:10/100/100/50/
```

Invalid Input/Output Window Specification!

An incorrect window specification has been setup using the "set-m2-run-window" command. The specification caused a disk file to be opened, instead of a window. A window specification must begin with "RAW:" or "CON:" and must be followed by the window dimensions.

Example:

```
RAW:0/0/320/100/
```

```
CON:10/100/100/50/
```

Insufficient Memory To Start User Process!

The system does not have sufficient memory to start user program. Since the program is already in memory the most likely reason for this error is that the user program stack is too large to fit into memory, if possible reduce the size of the stack using the "set-m2-run-stack-size" command.

Miscellaneous Errors

This subsection contains a description of the error messages produced by EMACS which are miscellaneous. Each error message is preceded with "ERROR:" to get the users attention.

Invalid Directory Path!

This error is displayed if the user has specified an invalid directory path in response to the prompt to change the current directory. A possible reason for this error is specifying a path for a volume which is not currently mounted.

Modula-2 Error Lister Error Messages

The error lister may generate a fatal error message if it has problems with opening or reading the error log file. An error message begins with "FATAL ERROR:" followed by a message.

Err Log File 'T:M2.err' Not Found!

The error log file could not be found in the "T:" directory. This error may occur if the "T:" assignment has not been set using the "ASSIGN" command or the file "M2.err" does not exist.

Err Log File 'T:M2.err' Is Invalid!

The error log file contains invalid data. This error should not occur normally. The solution is to erase the "M2.err" file and perform another compilation producing a valid error log file.

Modula-2 Object Module Generator

Most of the error messages produced by the object module generator are related to inconsistencies between the definition and implementation modules. Each error message is preceded with "ERROR:" to get the users attention.

Wrong Number of Parameters.

An incorrect number of parameters have been specified on the command line. Refer to the documentation for the object module generator for the proper command line syntax.

Input File '<FileName>.SBM' Could Not Be Opened.

The specified symbol file could not be opened. Usually indicates that the file does not exist in the specified directory.

Input File '<FileName>.COD' Could Not Be Opened.

The specified code file could not be opened. Usually indicates that the file does not exist in the specified directory.

Output File '<FileName>.OBM' Could Not Be Opened.

An output file could not be created. This error may result if the disk being written to is write protected or the disk has become full.

Input File '<FileName>.SBM' is Invalid.

The symbol file was found but contains invalid data. Generate a new symbol file and try the operation again.

Input File '<FileName>.COD' is Invalid.

The code file was found but contains invalid data. This error may result due to an error in the format of the file, caused by an improper assembly language program. Refer to the documentation on the assembly language interface for additional information.

SBM_Procedures < COD_Procedures.

The definition module defines fewer procedures than are declared in the implementation module. Examine the definition and implementation modules to resolve the conflict.

SBM_Procedures > COD_Procedures.

The definition module defines more procedures than are declared in the implementation module. Examine the definition and implementation modules to resolve the conflict.

SBM_Variables < COD_Variables.

The size of variables in the definition module is smaller than the mapping used in the implementation module for those variables.

SBM_Variables > COD_Variables.

The size of variables in the definition module is larger than the mapping used in the implementation module for those variables.

SBM_Modules < COD_Modules.

The number of modules imported in the definition module is inconsistent with the number of modules imported in the implementation module.

SBM_Modules > COD_Modules.

The number of modules imported in the definition module is inconsistent with the number of modules imported in the implementation module.

Input File '<FileName>.COD' does not have a 'hunk_end' after 'hunk_code'

After the hunk_code additional blocks of data were found. This is incorrect the code block should be followed by a hunk_end directive. Refer to the assembly language interface documentation for further information.

Internal. Memory Allocation Failure!

The object module generator has run out of memory. This can only happen if another application program has allocated a lot of memory. Remove any other tasks running from memory and then run the program again.

Appendix B	Command Summaries.....	3
Modula-2	Compiler.....	3
Modula-2	Linker.....	3
Modula-2	Error Lister.....	4
Modula-2	Compiler Configuration Utility.....	4
Modula-2	Symbol File Cross Reference Utility.....	4
Modula-2	Procedure Statistics Utility.....	4
Modula-2	Object Module Generator.....	4
	Quick Loader Utility.....	5
	EMACS Editor.....	5

EMACS Editor	2
Quick Loader Utility	3
Modula-2 Object Module Generator	4
Modula-2 Procedure Statement Utility	5
Modula-2 Symbol File Cross Reference Utility	6
Modula-2 Compiler Configuration Utility	7
Modula-2 Error Listener	8
Modula-2 Linker	9
Modula-2 Compiler	10
Appendix B Command Summaries	11

This appendix contains a quick reference summary of the command line usage and commands for each software tool. For additional information on a given command refer to the appropriate chapter of this manual.

Modula-2 Compiler

Command Line Usage:

M2 [Options] [SourceFile] [SourceFile] ...

Options:

-s <DirectoryPath>	Symbol File Search Path
-o <DirectoryPath>	Output File Directory
-r	Range Checking Toggle
-v	Overflow Checking Toggle
-g	Generate Reference File Toggle
-l	Generate 32-bit Addressing Toggle
-k	Version Control Key to Zero Toggle
-h <Size>	Heap Size
-i <Size>	Identifier Buffer Size
-c <Size>	Code Buffer Size
-a <Size>	Address Relocation Buffer Size
-b <Size>	Constant Buffer Size
-d <Size>	Disk Buffer Size

Modula-2 Linker

Command Line Usage:

M2Lk [Options] <MainModule> [MainModule] ...

Options:

-s <DirectoryPath>	Input Files Search Path
-o <DirectoryPath>	Output File Directory
-d	Generate Debugging Symbols
-a	Prefix Symbols With Module Name
-g	Generate Source Level Debug Info

Modula-2 Software Construction Set Reference Guide

Modula-2 Error Lister

Command Line Usage:

M2Err [Options]

Options:

-w Output Error List To A Window
-p Pause After Each Error

Modula-2 Compiler Configuration Utility

Command Line Usage:

M2Config [Options] <M2_FileName>

Options:

-d Reset To Default Configuration
-s Show Current Configuration With No Change

Modula-2 Symbol File Cross Reference Utility

Command Line Usage:

M2SymXRef <FileList> <OutputFile>

Modula-2 Procedure Statistics Utility

Command Line Usage:

M2Stat <ProgFileName> [ProgParameters]

Modula-2 Object Module Generator

Command Line Usage:

M2GenOBM [Options] <InputFileName> [OutputFileName]

Options:

-m Skip Imported Modules Check

Quick Loader Utility

Command Line Usage:

```
QLoad <QuickFile> <OutputPath>
QLoad -m <FileList> <QuickFile>
```

EMACS Editor

Command Line Usage:

```
M2Ed [FileName] [FileName] ...
```

Functions:

Cursor Commands

backward-char	[Ctrl-B]
backward-paragraph	[Esc-[]
backward-word	[Esc-B]
beginning-of-buffer	[Esc-<]
beginning-of-line	[Ctrl-A]
end-of-buffer	[Esc->]
end-of-line	[Ctrl-E]
forward-char	[Ctrl-F]
forward-paragraph	[Esc-]]
forward-word	[Esc-F]
goto-line	[Esc-G]
next-line	[Ctrl-N]
previous-line	[Ctrl-P]

Window Commands

delete-other-windows	[Ctrl-X 1]
delete-window	[Ctrl-X 0]
enlarge-window	[Ctrl-X ^]
next-window	[Ctrl-X N or Ctrl-X 0]
previous-window	[Ctrl-X P]
recenter	[Ctrl-L]
scroll-down	[Esc-V]
scroll-other-window	[Esc Ctrl-V]
scroll-up	[Ctrl-V]
shrink-window	
split-window-vertically	[Ctrl-X 2]

Buffer Commands

```
insert-buffer
```

Modula-2 Software Construction Set Reference Guide

kill-buffer
list-buffers
not-modified
switch-to-buffer
switch-to-buffer-other-window

[Ctrl-X K]
[Ctrl-X Ctrl-B]
[Esc-~]
[Ctrl-X B]
[Ctrl-X 4 B]

File Commands

current-dir
exact-fname-mode
file-req-mode
find-file
find-file-other-window
insert-file
make-backup-files
save-buffer
save-buffers-kill-emacs
save-some-buffers
write-file

[Ctrl-X Ctrl-F or F9]
[Ctrl-X 4 F or Shift-F9]
[Ctrl-X I]

[Ctrl-X Ctrl-S or F10]
[Ctrl-X Ctrl-C]
[Ctrl-X S]
[Ctrl-X Ctrl-W or Shift-F10]

Copy/Delete/Kill Commands

backward-delete-char
backward-kill-word
copy-region-as-kill
delete-blank-lines
delete-char
kill-line
kill-paragraph
kill-region
kill-word
yank

[Ctrl-H or BackSpace]
[Esc Backspace]
[Esc-W]
[Ctrl-X Ctrl-O]
[Ctrl-D or DEL]
[Ctrl-K]

[Ctrl-W]
[Esc-D or Esc DEL]
[Ctrl-Y]

Search/Replace Commands

isearch-backward
isearch-forward
query-replace
search-again
search-backward
search-forward

[Ctrl-R]
[Ctrl-S]
[Esc-%]
[Esc-A]
[Esc-R]
[Esc-S]

Case Commands

capitalize-word
downcase-region
downcase-word
upcase-region

[Esc-C]
[Ctrl-X Ctrl-L]
[Esc-L]
[Ctrl-X Ctrl-U]

upcase-word

[Esc-U]

Key Binding Commands

describe-bindings
describe-key-briefly
global-set-key
global-unset-key
help

[HELP]

Macro Commands

call-last-kbd-macro
end-kbd-macro
start-kbd-macro

[Ctrl-X E]
[Ctrl-X)]
[Ctrl-X (]

Fill Mode Commands

auto-fill-mode
fill-paragraph
insert-with-wrap
set-fill-column

[Esc-Q]
[Ctrl-X F]

Expression Commands

eval-current-buffer
eval-expression
load

Miscellaneous Commands

auto-indent-mode
blink-matching-paren
blink-matching-paren-hack
ctrlx-four-hack
delete-horizontal-space
exchange-point-and-mark
execute-extended-command
free-mem-avail
insert-newline
just-one-space
keyboard-quit
newline-and-indent
open-line
overwrite-mode
prefix-region
redraw-display

[Ctrl-X 4]
[Ctrl-\]
[Ctrl-X Ctrl-X]
[Esc-X]
[Shift-F5]
[Return]
[Esc-Space]
[Ctrl-G]
[Ctrl-J]
[Ctrl-O]

Modula-2 Software Construction Set Reference Guide

quoted-insert	[Ctrl-Q]
self-insert-command	
set-mark-command	[Ctrl-@]
set-prefix-string	
suspend-emacs	[Ctrl-Z]
transpose-chars	[Ctrl-T]
what-cursor-position	[Ctrl-X =]

Mouse Commands

amiga-mouse
mouse-recenter
mouse-kill-word
mouse-kill-line
mouse-delete-char
mouse-just-one-space
mouse-kill-region
mouse-yank
mouse-scroll-up
mouse-scroll-down
mouse-split-window-vertically
mouse-delete-window
mouse-beginning-of-buffer
mouse-end-of-buffer
mouse-enlarge-window
mouse-shrink-window

Display Setup Commands

set-mode-background
set-mode-foreground
set-mode-rendition
set-text-background
set-text-foreground
set-text-rendition
toggle-window-hack

Modula-2 Compiler Commands

m2-comp-load	[F8]
m2-comp-unload	[Shift-F8]
m2-comp-current-buffer	[F2]
m2-comp-buffer	[Shift-F2]
m2-comp-file	[F5]
m2-err-current-error	[F1-Shift]
m2-err-next-error	[F1]
set-m2-comp-opt	
set-m2-comp-stack-size	
set-m2-err-count	

Modula-2 Linker Commands

Command	Shift	Key
m2-link-main	[F3]	Left
m2-link-module	[Shift-F3]	Left
set-m2-link-out-dir	[Shift-F6]	Right
set-m2-link-gen-sym	[Shift-F7]	Right
set-m2-main-module	[F6]	Up
set-m2-obj-dir	[F7]	Up
		Down
		Down

Modula-2 Run Commands

m2-run-main	[F4]
m2-run-module	[Shift-F4]

set-m2-run-args
set-m2-run-stack-size
set-m2-run-window

Command	Shift	Key
m2-err-next-error		F1
m2-err-current-error	*	F1
m2-comp-current-buffer		F2
m2-comp-buffer	*	F2
m2-link-main		F3
m2-link-module	*	F3
m2-run-main		F4
m2-run-module	*	F4
m2-comp-file		F5
m2-run-args	*	F5
set-m2-main-module		F6
set-m2-link-out-dir	*	F6
set-m2-obj-dir		F7
set-m2-link-gen-sym	*	F7
m2-comp-load		F8
m2-comp-unload	*	F8
find-file		F9
find-file-other-window	*	F9
save-buffer		F10
write-file	*	F10

Arrow Key Bindings

Key	Shift	Command
Left		backward-char
Left	*	backward-word
Right		forward-char
Right	*	forward-word
Up		previous-line
Up	*	backward-paragraph
Down		next-line
Down	*	forward-paragraph

Function Key Bindings

Key	Shift	Command
F1		m2-err-next-error
F1	*	m2-err-current-error
F2		m2-comp-current-buffer
F2	*	m2-comp-buffer
F3		m2-link-main
F3	*	m2-link-module
F4		m2-run-main
F4	*	m2-run-module
F5		m2-comp-file
F5	*	free-mem-avail
F6		set-m2-main-module
F6	*	set-m2-link-out-dir
F7		set-m2-obj-dir
F7	*	set-m2-link-gen-sym
F8		m2-comp-load
F8	*	m2-comp-unload
F9		find-file
F9	*	find-file-other-window
F10		save-buffer
F10	*	write-file

Mouse Keyboard Combination Bindings

Text Area Commands

Ctrl	Alt	Shift	Command
		*	amiga-mouse
	*		mouse-recenter
	*	*	mouse-kill-word
*			mouse-kill-line
*		*	mouse-delete-char
*	*		mouse-just-one-space
*	*	*	mouse-kill-region
*	*	*	mouse-yank

Mode Line Commands

Ctrl	Alt	Shift	Command
		*	mouse-scroll-up
	*		mouse-scroll-down
	*	*	mouse-split-window-vertically
*			mouse-delete-window
*		*	mouse-beginning-of-buffer
*	*		mouse-end-of-buffer
*	*	*	mouse-enlarge-window
*	*	*	mouse-shrink-window

Echo Line Commands

Ctrl	Alt	Shift	Command
		*	switch-to-buffer
	*		kill-buffer
	*	*	describe-key-briefly
*			describe-bindings
*		*	suspend-emacs
*	*		save-buffers-kill-emacs
*	*	*	list-buffers
*	*	*	toggle-window-hack

Appendix C Definition Modules Cross Reference.....3

Appendix C Definition Modules Cross Reference.....3

Definition Modules Cross Reference

This appendix contains a cross reference of the identifiers defined in the definition modules. In addition, the class(CONST, TYPE, VAR or PROCEDURE) of each identifier is shown in the right column of the cross reference.

Identifier	Module	Type
ABC	Blit	CONST
AbleICR	CIAResource	PROCEDURE
aBMS	PrinterDevice	CONST
ABNC	Blit	CONST
AbortIO	IODevices	PROCEDURE
aCAM	PrinterDevice	CONST
AccessRead	AmigaDOS	CONST
AccessWrite	AmigaDOS	CONST
ACK	ASCII	CONST
ActionCopyDir	AmigaDOSProcess	CONST
ActionCreateDir	AmigaDOSProcess	CONST
ActionCurrentVolume	AmigaDOSProcess	CONST
ActionDeleteObject	AmigaDOSProcess	CONST
ActionDie	AmigaDOSProcess	CONST
ActionDiskChange	AmigaDOSProcess	CONST
ActionDiskInfo	AmigaDOSProcess	CONST
ActionDiskType	AmigaDOSProcess	CONST
ActionEvent	AmigaDOSProcess	CONST
ActionExamineNext	AmigaDOSProcess	CONST
ActionExamineObject	AmigaDOSProcess	CONST
ActionFreeLock	AmigaDOSProcess	CONST
ActionGetBlock	AmigaDOSProcess	CONST
ActionInfo	AmigaDOSProcess	CONST
ActionInhibit	AmigaDOSProcess	CONST
ActionLocateObject	AmigaDOSProcess	CONST
ActionNIL	AmigaDOSProcess	CONST
ActionParent	AmigaDOSProcess	CONST
ActionRead	AmigaDOSProcess	CONST
ActionRenameDisk	AmigaDOSProcess	CONST
ActionRenameObject	AmigaDOSProcess	CONST
ActionSetComment	AmigaDOSProcess	CONST
ActionSetFileDate	AmigaDOSProcess	CONST
ActionSetMap	AmigaDOSProcess	CONST
ActionSetProtect	AmigaDOSProcess	CONST
ActionSetRawMode	AmigaDOSProcess	CONST
ActionTimer	AmigaDOSProcess	CONST
ActionWaitChar	AmigaDOSProcess	CONST
ActionWrite	AmigaDOSProcess	CONST
Activate	Intuition	CONST
ActivateGadget	Intuition	PROCEDURE
ActivateWindow	Intuition	PROCEDURE
ActiveWindow	Intuition	CONST
ADAllocMaxPrec	AudioDevice	CONST
ADAllocMinPrec	AudioDevice	CONST
ADCmdAllocate	AudioDevice	CONST

Modula-2 Software Construction Set Reference Guide

ADCmdFinish	AudioDevice	CONST
ADCmdFree	AudioDevice	CONST
ADCmdLock	AudioDevice	CONST
ADCmdNoUnit	AudioDevice	CONST
ADCmdPerVol	AudioDevice	CONST
ADCmdSetPrec	AudioDevice	CONST
ADCmdWaitCycle	AudioDevice	CONST
AddAnimOb	Gels	PROCEDURE
AddBob	Gels	PROCEDURE
AddConfigDev	Expansion	PROCEDURE
AddDevice	IODevices	PROCEDURE
AddDosNode	Expansion	PROCEDURE
AddFont	Text	PROCEDURE
AddFreeList	Workbench	PROCEDURE
AddGadget	Intuition	PROCEDURE
AddGList	Intuition	PROCEDURE
AddHead	Lists	PROCEDURE
AddICRVector	CIAResource	PROCEDURE
AddIntServer	Interrupts	PROCEDURE
AddLibrary	Libraries	PROCEDURE
AddMemList	Memory	PROCEDURE
AddPort	Ports	PROCEDURE
AddResource	Resources	PROCEDURE
AddSemaphore	Semaphores	PROCEDURE
AddTail	Lists	PROCEDURE
AddTask	Tasks	PROCEDURE
AddTime	TimerDevice	PROCEDURE
AddVSprite	Gels	PROCEDURE
aDEN1	PrinterDevice	CONST
aDEN2	PrinterDevice	CONST
aDEN3	PrinterDevice	CONST
aDEN4	PrinterDevice	CONST
aDEN5	PrinterDevice	CONST
aDEN6	PrinterDevice	CONST
ADHardChannels	AudioDevice	CONST
ADIOErrAllocFailed	AudioDevice	CONST
ADIOErrChannelStolen	AudioDevice	CONST
ADIOErrNoAllocation	AudioDevice	CONST
ADIONoWait	AudioDevice	CONST
ADIOPerVol	AudioDevice	CONST
ADIOSyncCycle	AudioDevice	CONST
ADIOWriteMessage	AudioDevice	CONST
ADKFast	ADKHardware	CONST
ADKMFMPPrec	ADKHardware	CONST
ADKMSBSync	ADKHardware	CONST
ADKPre000NS	ADKHardware	CONST
ADKPre140NS	ADKHardware	CONST
ADKPre280NS	ADKHardware	CONST
ADKPre560NS	ADKHardware	CONST
ADKPreComp0	ADKHardware	CONST
ADKPreComp1	ADKHardware	CONST
ADKSetClr	ADKHardware	CONST

Definition Modules Cross Reference

ADKUARTBrk	ADKHardware	CONST
ADKUse0P1	ADKHardware	CONST
ADKUse0V1	ADKHardware	CONST
ADKUse1P2	ADKHardware	CONST
ADKUse1V2	ADKHardware	CONST
ADKUse2P3	ADKHardware	CONST
ADKUse2V3	ADKHardware	CONST
ADKUse3PN	ADKHardware	CONST
ADKUse3VN	ADKHardware	CONST
ADKWordSync	ADKHardware	CONST
ADNStartProc	Expansion	CONST
aEXTEND	PrinterDevice	CONST
AF10	DiskFont	CONST
AF11	DiskFont	CONST
AF12	DiskFont	CONST
AF13	DiskFont	CONST
AF14	DiskFont	CONST
AF15	DiskFont	CONST
AF2	DiskFont	CONST
AF3	DiskFont	CONST
AF4	DiskFont	CONST
AF5	DiskFont	CONST
AF6	DiskFont	CONST
AF68010	ExecBase	CONST
AF68020	ExecBase	CONST
AF68881	ExecBase	CONST
AF7	DiskFont	CONST
AF8	DiskFont	CONST
AF9	DiskFont	CONST
AFDisk	DiskFont	CONST
AFMemory	DiskFont	CONST
aFNT0	PrinterDevice	CONST
aFNT1	PrinterDevice	CONST
aFNT10	PrinterDevice	CONST
aFNT2	PrinterDevice	CONST
aFNT3	PrinterDevice	CONST
aFNT4	PrinterDevice	CONST
aFNT5	PrinterDevice	CONST
aFNT6	PrinterDevice	CONST
aFNT7	PrinterDevice	CONST
aFNT8	PrinterDevice	CONST
aFNT9	PrinterDevice	CONST
AGIOError	Alerts	CONST
AGMakeLib	Alerts	CONST
AGNoMemory	Alerts	CONST
AGNoSignal	Alerts	CONST
AGOpenDev	Alerts	CONST
AGOpenLib	Alerts	CONST
AGOpenRes	Alerts	CONST
aHTS	PrinterDevice	CONST
aIND	PrinterDevice	CONST
aJFY0	PrinterDevice	CONST

Modula-2 Software Construction Set Reference Guide

aJFY1	PrinterDevice	CONST
aJFY3	PrinterDevice	CONST
aJFY5	PrinterDevice	CONST
aJFY6	PrinterDevice	CONST
aJFY7	PrinterDevice	CONST
Alert	Alerts	PROCEDURE
AlertLayersNoMem	Clipping	CONST
AlertType	Intuition	CONST
AllocAbs	Memory	PROCEDURE
ALLOCATE	Heap	PROCEDURE
Allocate	Memory	PROCEDURE
ALLOCATE	Storage	PROCEDURE
AllocBoardMem	Expansion	PROCEDURE
AllocCList	CList	PROCEDURE
AllocConfigDev	Expansion	PROCEDURE
AllocEntry	Memory	PROCEDURE
AllocExpansionMem	Expansion	PROCEDURE
AllocMem	Memory	PROCEDURE
AllocPotBits	PotgoResource	PROCEDURE
AllocRaster	Rasters	PROCEDURE
AllocRemember	Intuition	PROCEDURE
AllocSignal	Tasks	PROCEDURE
AllocTrap	Tasks	PROCEDURE
aLMS	PrinterDevice	CONST
AlohaWorkbench	IntuitionBase	PROCEDURE
AlphabetChars	ASCII	CONST
AlphaP101	Preferences	CONST
AltKeyMap	Intuition	CONST
AltLeft	Intuition	CONST
AltRight	Intuition	CONST
AmigaKeys	Intuition	CONST
AmigaLeft	Intuition	CONST
AmigaRight	Intuition	CONST
ANAddSWGadget	Alerts	CONST
ANAsyncPkt	Alerts	CONST
ANAudioDev	Alerts	CONST
ANBadChkSum	Alerts	CONST
ANBadExpansionFree	Alerts	CONST
ANBadGadget	Alerts	CONST
ANBadMessage	Alerts	CONST
ANBadOverlay	Alerts	CONST
ANBadSegList	Alerts	CONST
ANBadState	Alerts	CONST
ANBaseChkSum	Alerts	CONST
ANBC	Blit	CONST
ANBitMap	Alerts	CONST
ANBltBitMap	Alerts	CONST
ANBNC	Blit	CONST
ANBogusExcpt	Alerts	CONST
ANBootError	Alerts	CONST
ANBootStrap	Alerts	CONST
ANCIARsrc	Alerts	CONST

Definition Modules Cross Reference

ANCListLib	Alerts	CONST
ANConsoleDev	Alerts	CONST
ANCreatePort	Alerts	CONST
ANDiskBlkSeq	Alerts	CONST
ANDiskCopy	Alerts	CONST
ANDiskError	Alerts	CONST
ANDiskRsrc	Alerts	CONST
ANDOSLib	Alerts	CONST
AndRectRegion	Regions	PROCEDURE
AndRegionRegion	Regions	PROCEDURE
ANDRHasDisk	Alerts	CONST
ANDRIntNoAct	Alerts	CONST
aNEL	PrinterDevice	CONST
ANEndTask	Alerts	CONST
ANExcptVect	Alerts	CONST
ANExecLib	Alerts	CONST
ANExpansionLib	Alerts	CONST
AnFracSize	Gels	CONST
ANFreeTwice	Alerts	CONST
ANFreeVec	Alerts	CONST
ANGadgetType	Alerts	CONST
ANGamePortDev	Alerts	CONST
ANGfxNoLCM	Alerts	CONST
ANGfxNoMem	Alerts	CONST
ANGraphicsLib	Alerts	CONST
ANIconLib	Alerts	CONST
Animate	Gels	PROCEDURE
AnimComp	Gels	TYPE
AnimCompProc	Gels	TYPE
AnimCompPtr	Gels	TYPE
AnimHalf	Gels	CONST
AnimOb	Gels	TYPE
AnimObProc	Gels	TYPE
AnimObPtr	Gels	TYPE
ANInitAPtr	Alerts	CONST
ANIntrMem	Alerts	CONST
ANIntuition	Alerts	CONST
ANItemAlloc	Alerts	CONST
ANItemBoxTop	Alerts	CONST
ANKeyboardDev	Alerts	CONST
ANKeyFree	Alerts	CONST
ANKeyRange	Alerts	CONST
ANLayersLib	Alerts	CONST
ANLayersNoMem	Alerts	CONST
ANLibChkSum	Alerts	CONST
ANLibMem	Alerts	CONST
ANLongFrame	Alerts	CONST
ANMakeVPort	Alerts	CONST
ANMathLib	Alerts	CONST
ANMemCorrupt	Alerts	CONST
ANMiscRsrc	Alerts	CONST
ANNoConsole	Alerts	CONST

Modula-2 Software Construction Set Reference Guide

ANOpenScreen	Alerts	CONST
ANOpenScrnRast	Alerts	CONST
ANOpenWindow	Alerts	CONST
ANPlaneAlloc	Alerts	CONST
ANQPktFail	Alerts	CONST
ANRAMLib	Alerts	CONST
ANRegionMemory	Alerts	CONST
ANSemCorrupt	Alerts	CONST
ANShortFrame	Alerts	CONST
ANStartMem	Alerts	CONST
ANSubAlloc	Alerts	CONST
ANSysScrnType	Alerts	CONST
ANTDCalibSeek	Alerts	CONST
ANTDDelay	Alerts	CONST
ANTextTmpRas	Alerts	CONST
ANTimerDev	Alerts	CONST
ANTMBadReq	Alerts	CONST
ANTMBadSupply	Alerts	CONST
ANTrackDiskDev	Alerts	CONST
ANWeirdEcho	Alerts	CONST
ANWorkbench	Alerts	CONST
AnySignal	Tasks	CONST
AnySprite	Sprites	CONST
AnyTrap	Tasks	CONST
AOAudioDev	Alerts	CONST
AOBootStrap	Alerts	CONST
AOCIARsrc	Alerts	CONST
AOCListLib	Alerts	CONST
AOConsoleDev	Alerts	CONST
AODiskRsrc	Alerts	CONST
AODOSLib	Alerts	CONST
AOExecLib	Alerts	CONST
AOExpansionLib	Alerts	CONST
AOGamePortDev	Alerts	CONST
AOGraphicsLib	Alerts	CONST
AOIconLib	Alerts	CONST
AOIntuition	Alerts	CONST
AOKeyboardDev	Alerts	CONST
AOLayersLib	Alerts	CONST
AOMathLib	Alerts	CONST
AOMiscRsrc	Alerts	CONST
AORAMLib	Alerts	CONST
AorB	Blit	CONST
AorC	Blit	CONST
AOTimerDev	Alerts	CONST
AOTrackDiskDev	Alerts	CONST
AOWorkbench	Alerts	CONST
aPERF	PrinterDevice	CONST
aPERFØ	PrinterDevice	CONST
aPLD	PrinterDevice	CONST
aPLU	PrinterDevice	CONST
aPROPØ	PrinterDevice	CONST

Definition Modules Cross Reference

aPROP1	CONST	PrinterDevice	CONST
aPROP2	CONST	PrinterDevice	CONST
arccos	CONST	MathLib0	PROCEDURE
arcsin	CONST	MathLib0	PROCEDURE
arctan	CONST	MathLib0	PROCEDURE
AreaCircle	CONST	Areas	PROCEDURE
AreaDraw	CONST	Areas	PROCEDURE
AreaEllipse	PROCEDURE	Areas	PROCEDURE
AreaEnd	PROCEDURE	Areas	PROCEDURE
AreaInfo	TYPE	Areas	TYPE
AreaInfoPtr	TYPE	Areas	TYPE
AreaMove	TYPE	Areas	PROCEDURE
AreaOutline	CONST	Rasters	CONST
argc	CONST	System	VAR
argv	TYPE	System	VAR
aRI	CONST	PrinterDevice	CONST
aRIN	CONST	PrinterDevice	CONST
aRIS	CONST	PrinterDevice	CONST
aRMS	CONST	PrinterDevice	CONST
aSBC	CONST	PrinterDevice	CONST
aSFC	CONST	PrinterDevice	CONST
aSGR0	CONST	PrinterDevice	CONST
aSGR1	PROCEDURE	PrinterDevice	CONST
aSGR22	CONST	PrinterDevice	CONST
aSGR23	PROCEDURE	PrinterDevice	CONST
aSGR24	PROCEDURE	PrinterDevice	CONST
aSGR3	TYPE	PrinterDevice	CONST
aSGR4	TYPE	PrinterDevice	CONST
AShiftShift	PROCEDURE	Blit	CONST
aSHORP0	TYPE	PrinterDevice	CONST
aSHORP1	TYPE	PrinterDevice	CONST
aSHORP2	TYPE	PrinterDevice	CONST
aSHORP3	TYPE	PrinterDevice	CONST
aSHORP4	PROCEDURE	PrinterDevice	CONST
aSHORP5	CONST	PrinterDevice	CONST
aSHORP6	CONST	PrinterDevice	CONST
AskFont	CONST	Text	CONST
AskSoftStyle	CONST	Text	PROCEDURE
aSLPP	CONST	PrinterDevice	PROCEDURE
aSLRM	CONST	PrinterDevice	CONST
AspectHoriz	CONST	Preferences	CONST
AspectVert	CONST	Preferences	CONST
aSTBM	CONST	PrinterDevice	CONST
aSUS0	CONST	PrinterDevice	CONST
aSUS1	CONST	PrinterDevice	CONST
aSUS2	CONST	PrinterDevice	CONST
aSUS3	CONST	PrinterDevice	CONST
aSUS4	CONST	PrinterDevice	CONST
aTBC0	CONST	PrinterDevice	CONST
aTBC1	CONST	PrinterDevice	CONST
aTBC3	CONST	PrinterDevice	CONST
aTBC4	CONST	PrinterDevice	CONST

Modula-2 Software Construction Set Reference Guide

aTBCALL	PrinterDevice	CONST
aTBSALL	PrinterDevice	CONST
ATDeadEnd	Alerts	CONST
aTMS	PrinterDevice	CONST
AtoD	Blit	CONST
ATRecovery	Alerts	CONST
aTSS	PrinterDevice	CONST
AttemptLockLayerRom	RomLayers	PROCEDURE
AttemptSemaphore	Semaphores	PROCEDURE
AudChannel	CustomHardware	TYPE
AudChannelPtr	CustomHardware	TYPE
AudioChannelsSet	AudioDevice	TYPE
AudioName	AudioDevice	CONST
AUL	Blit	CONST
AUserStuff	Gels	TYPE
AutoBackPen	Intuition	CONST
AutoDrawMode	Intuition	CONST
AutoFrontPen	Intuition	CONST
AutoITextFont	Intuition	CONST
AutoKnob	Intuition	CONST
AutoLeftEdge	Intuition	CONST
AutoNextText	Intuition	CONST
AutoRequest	Intuition	PROCEDURE
AutoTopEdge	Intuition	CONST
Available	Heap	PROCEDURE
Available	Storage	PROCEDURE
AvailFont	DiskFont	TYPE
AvailFontPtr	DiskFont	TYPE
AvailFonts	DiskFont	PROCEDURE
AvailFontsHeader	DiskFont	TYPE
AvailFontsHeaderPtr	DiskFont	TYPE
AvailFontType	DiskFont	TYPE
AvailFontTypeSet	DiskFont	TYPE
AvailMem	Memory	PROCEDURE
aVERP0	PrinterDevice	CONST
aVERP1	PrinterDevice	CONST
aVTS	PrinterDevice	CONST
AxorC	Blit	CONST
B2Bobber	Gels	CONST
B2Norm	Gels	CONST
B2Swap	Gels	CONST
BackDrop	Intuition	CONST
BackSaved	Gels	CONST
Baud110	Preferences	CONST
Baud1200	Preferences	CONST
Baud19200	Preferences	CONST
Baud2400	Preferences	CONST
Baud300	Preferences	CONST
Baud4800	Preferences	CONST
Baud9600	Preferences	CONST
BaudMIDI	Preferences	CONST
BBNameDos	BootBlock	CONST

BBNameKick	BootBlock	CONST
BDrawn	Gels	CONST
Beeping	Intuition	CONST
BeginIO	IODevices	PROCEDURE
BeginRefresh	Intuition	PROCEDURE
BeginUpdate	Layers	PROCEDURE
BehindLayer	Layers	PROCEDURE
BEL	ASCII	CONST
BF14	Gels	CONST
BF15	Gels	CONST
BF2	Gels	CONST
BF3	Gels	CONST
BF4	Gels	CONST
BF5	Gels	CONST
BF6	Gels	CONST
BF7	Gels	CONST
BIF1	Intuition	CONST
BIF10	Intuition	CONST
BIF11	Intuition	CONST
BIF12	Intuition	CONST
BIF13	Intuition	CONST
BIF14	Intuition	CONST
BIF15	Intuition	CONST
BIF2	Intuition	CONST
BIF3	Intuition	CONST
BIF4	Intuition	CONST
BIF5	Intuition	CONST
BIF6	Intuition	CONST
BIF7	Intuition	CONST
BIF8	Intuition	CONST
BIF9	Intuition	CONST
BitMap	Graphics	TYPE
BitMapPtr	Graphics	TYPE
BlitMsgFault	GraphicsBase	CONST
BlitReverse	Blit	CONST
BltBitMap	Blit	PROCEDURE
BltBitMapRastPort	Blit	PROCEDURE
BltClear	Blit	PROCEDURE
BltClearFlagsSet	Blit	TYPE
BltClearWait	Blit	CONST
BltClearXY	Blit	CONST
BltMaskBitMapRastPort	Blit	PROCEDURE
bltnode	Blit	TYPE
bltnodeptr	Blit	TYPE
BltPattern	Blit	PROCEDURE
BltTemplate	Blit	PROCEDURE
BndryOff	Drawing	PROCEDURE
Bob	Gels	TYPE
BobFlags	Gels	TYPE
BobFlagsSet	Gels	TYPE
BobIsComp	Gels	CONST
BobNix	Gels	CONST

Modula-2 Software Construction Set Reference Guide

BobPtr	Gels	TYPE
BobsAway	Gels	CONST
BobUpdate	Gels	CONST
Bold	Text	CONST
BoolExtend	Intuition	CONST
BoolGadget	Intuition	CONST
BoolInfo	Intuition	TYPE
BoolInfoFlags	Intuition	TYPE
BoolInfoFlagsSet	Intuition	TYPE
BoolInfoPtr	Intuition	TYPE
BoolMask	Intuition	CONST
BootBlock	BootBlock	TYPE
BootBlockPtr	BootBlock	TYPE
BootSects	BootBlock	CONST
Border	Intuition	TYPE
BorderHit	Gels	CONST
Borderless	Intuition	CONST
BorderPtr	Intuition	TYPE
BottomBorder	Intuition	CONST
BottomHit	Gels	CONST
BoundaryColMask	Gels	TYPE
BoundaryColProc	Gels	TYPE
BPTR	AmigaDOS	TYPE
Brother15XL	Preferences	CONST
BS	ASCII	CONST
BShiftShift	Blit	CONST
BSTR	AmigaDOS	TYPE
BufferSize	FileSystem	VAR
BuildSysRequest	Intuition	PROCEDURE
BumpRevision	Workbench	PROCEDURE
BUserFlags	Gels	CONST
BUserStuff	Gels	TYPE
BusyRead	Terminal	PROCEDURE
BWaiting	Gels	CONST
CAN	ASCII	CONST
Cause	Interrupts	PROCEDURE
CBDCurrentReadID	ClipboardDevice	CONST
CBDCurrentWriteID	ClipboardDevice	CONST
CBDPPost	ClipboardDevice	CONST
CBErrObsoleteID	ClipboardDevice	CONST
CBMMPS1000	Preferences	CONST
CBump	Copper	PROCEDURE
CDAskDefaultKeyMap	ConsoleDevice	CONST
CDAskKeyMap	ConsoleDevice	CONST
CDConfigMe	Expansion	CONST
CDSetDefaultKeyMap	ConsoleDevice	CONST
CDSetKeyMap	ConsoleDevice	CONST
CDShutup	Expansion	CONST
CEND	CopperUtil	PROCEDURE
ChangeSprite	Sprites	PROCEDURE
CharCases	ASCII	TYPE
CharIsASCII	ASCII	PROCEDURE

Definition Modules Cross Reference

CharIsControl	ASCII	PROCEDURE
CharIsPrintable	ASCII	PROCEDURE
Checked	Intuition	CONST
CheckIO	IODevices	PROCEDURE
CheckIt	Intuition	CONST
CheckWidth	Intuition	CONST
CIA	CIAHardware	TYPE
ciaa	CIAHardware	VAR
CIAAName	CIAResource	CONST
ciab	CIAHardware	VAR
CIABITS	CIAHardware	TYPE
CIABName	CIAResource	CONST
CIAComCD	CIAHardware	CONST
CIAComCTS	CIAHardware	CONST
CIAComDSR	CIAHardware	CONST
CIAComDTR	CIAHardware	CONST
CIAComRTS	CIAHardware	CONST
CIAcraINMODE	CIAHardware	CONST
CIAcraLOAD	CIAHardware	CONST
CIAcraOUTMODE	CIAHardware	CONST
CIAcraPBON	CIAHardware	CONST
CIAcraRUNMODE	CIAHardware	CONST
CIAcraSPMODE	CIAHardware	CONST
CIAcraSTART	CIAHardware	CONST
CIAcraTODIN	CIAHardware	CONST
CIAcrbALARM	CIAHardware	CONST
CIAcrbInCNT	CIAHardware	CONST
CIAcrbInCNTTA	CIAHardware	CONST
CIAcrbINMODE0	CIAHardware	CONST
CIAcrbINMODE1	CIAHardware	CONST
CIAcrbInPHI2	CIAHardware	CONST
CIAcrbInTA	CIAHardware	CONST
CIAcrbLOAD	CIAHardware	CONST
CIAcrbOUTMODE	CIAHardware	CONST
CIAcrbPBON	CIAHardware	CONST
CIAcrbRUNMODE	CIAHardware	CONST
CIAcrbSTART	CIAHardware	CONST
CIADskChange	CIAHardware	CONST
CIADskDIREC	CIAHardware	CONST
CIADskMOTOR	CIAHardware	CONST
CIADskProt	CIAHardware	CONST
CIADskRdy	CIAHardware	CONST
CIADskSEL0	CIAHardware	CONST
CIADskSEL1	CIAHardware	CONST
CIADskSEL2	CIAHardware	CONST
CIADskSEL3	CIAHardware	CONST
CIADskSIDE	CIAHardware	CONST
CIADskSTEP	CIAHardware	CONST
CIADskTrack0	CIAHardware	CONST
CIAGamePort0	CIAHardware	CONST
CIAGamePort1	CIAHardware	CONST
CIAicrALRM	CIAHardware	CONST

Modula-2 Software Construction Set Reference Guide

CIAicrFLG	CIAHardware	CONST
CIAicrIR	CIAHardware	CONST
CIAicrSETCLR	CIAHardware	CONST
CIAicrSP	CIAHardware	CONST
CIAicrTA	CIAHardware	CONST
CIAicrTB	CIAHardware	CONST
CIALED	CIAHardware	CONST
CIAOverlay	CIAHardware	CONST
CIAPAD	CIAHardware	TYPE
CIAPrtrBUSY	CIAHardware	CONST
CIAPrtrPOUT	CIAHardware	CONST
CIAPrtrSEL	CIAHardware	CONST
CIAPtr	CIAHardware	TYPE
CINIT	CopperUtil	PROCEDURE
CleanMe	Blit	CONST
CleanUp	Blit	CONST
ClearDMRequest	Intuition	PROCEDURE
ClearEOL	Drawing	PROCEDURE
ClearMenuStrip	Intuition	PROCEDURE
ClearPointer	Intuition	PROCEDURE
ClearRectRegion	Regions	PROCEDURE
ClearRegion	Regions	PROCEDURE
ClearScreen	Drawing	PROCEDURE
ClipBlit	Blit	PROCEDURE
ClipboardUnitPartial	ClipboardDevice	TYPE
ClipboardUnitPartialPtr	ClipboardDevice	TYPE
ClipRect	Clipping	TYPE
ClipRectPtr	Clipping	TYPE
CLIReturnCode	System	VAR
CListBase	CList	VAR
CListDesc	CList	TYPE
CListName	CList	CONST
CListPool	CList	TYPE
Close	AmigaDOS	PROCEDURE
Close	FileSystem	PROCEDURE
Close	Intuition	CONST
CloseDevice	IODevices	PROCEDURE
CloseFont	Text	PROCEDURE
CloseInput	InOut	PROCEDURE
CloseLibrary	Libraries	PROCEDURE
CloseMathLib0	InitMathLib0	PROCEDURE
CloseOutput	InOut	PROCEDURE
CloseScreen	Intuition	PROCEDURE
CloseWindow	Intuition	CONST
CloseWindow	Intuition	PROCEDURE
CloseWorkBench	Intuition	PROCEDURE
CmdClear	IODevices	CONST
CmdFlush	IODevices	CONST
CmdInvalid	IODevices	CONST
CmdLineLength	System	VAR
CmdLinePtr	System	VAR
CmdNonStd	IODevices	CONST

Definition Modules Cross Reference

CmdRead	IODevices	CONST
CmdReset	IODevices	CONST
CmdStart	IODevices	CONST
CmdStop	IODevices	CONST
CmdUpdate	IODevices	CONST
CmdWrite	IODevices	CONST
CMove	Copper	PROCEDURE
CMOVE	CopperUtil	PROCEDURE
CmpTime	TimerDevice	PROCEDURE
collTable	Gels	TYPE
collTablePtr	Gels	TYPE
ColorMap	Views	TYPE
ColorMapPtr	Views	TYPE
ColorOn	Display	CONST
ColorTable	Views	TYPE
ColorTablePtr	Views	TYPE
CommandLineInterface	AmigaDOSExt	TYPE
CommandLineInterfacePtr	AmigaDOSExt	TYPE
CommSeq	Intuition	CONST
CommWidth	Intuition	CONST
CompareString	Strings	PROCEDURE
CompareStringCAP	Strings	PROCEDURE
Complement	Rasters	CONST
ConcatCList	CList	PROCEDURE
ConcatString	Strings	PROCEDURE
ConfigBoard	Expansion	PROCEDURE
ConfigChain	Expansion	PROCEDURE
ConfigDev	Expansion	PROCEDURE
ConfigDevPtr	Expansion	TYPE
ConsoleBase	ConsoleDevice	TYPE
ConsoleName	ConsoleDevice	VAR
Control	ASCII	CONST
ControlShift	ASCII	CONST
ConUnit	ConsoleDevUnit	CONST
ConUnitPtr	ConsoleDevUnit	TYPE
ConvNumberToString	Conversions	TYPE
ConvRealToString	RealConversions	PROCEDURE
ConvStringToNumber	Conversions	PROCEDURE
ConvStringToReal	RealConversions	PROCEDURE
ConvStringToUpperCase	Strings	PROCEDURE
copinit	Copper	TYPE
copinitPtr	Copper	TYPE
CopIns	Copper	TYPE
CopInsPtr	Copper	TYPE
CopList	Copper	TYPE
CopListPtr	Copper	TYPE
CopperMove	Copper	TYPE
CopperWait	Copper	CONST
CopyCList	Copper	CONST
CopyMem	CList	PROCEDURE
CopyMemQuick	Memory	PROCEDURE
CopySBitMap	Memory	PROCEDURE
	RomLayers	PROCEDURE

Modula-2 Software Construction Set Reference Guide

CopySet
 CopyString
 cos
 cosh
 cprlist
 cprlistPtr
 CprNtLof
 CprNtSht
 CprNxtBuf
 CR
 CreateBehindLayer
 CreateDir
 CreateExtIO
 CreatePort
 CreateProc
 CreateSet
 CreateStdIO
 CreateTask
 CreateUpfrontLayer
 CRNeedsNoConcealedRasters
 CTCHClrTab
 CTCHClrTabsAll
 CTCHSetTab
 CurrentBinding
 CurrentBindingPtr
 CurrentDir
 CurrentTask
 CurrentTime
 CursorDown
 CursorLeft
 CursorRight
 CursorUp
 Custom
 custom
 Custom
 CustomBitmap
 CustomName
 CustomPtr
 CustomScreen
 CWait
 CWAIT
 DACBindTime
 DACBootTime
 DACBusWidth
 DACByteWide
 DACConfigTime
 DACNever
 DACNibbleWide
 DACWordWide
 DateStamp
 DateStampPtr
 DateStampRecord

LargeSets
 Strings
 MathLib0
 MathLib0
 Copper
 Copper
 Copper
 Copper
 Copper
 ASCII
 Layers
 AmigaDOS
 IODevicesUtil
 PortsUtil
 AmigaDOSProcess
 LargeSets
 IODevicesUtil
 TasksUtil
 Layers
 Clipping
 ConsoleDevice
 ConsoleDevice
 ConsoleDevice
 Expansion
 Expansion
 AmigaDOS
 Tasks
 Intuition
 Intuition
 Intuition
 Intuition
 Intuition
 CustomHardware
 CustomHardware
 Preferences
 Intuition
 Preferences
 CustomHardware
 Intuition
 Copper
 CopperUtil
 ConfigRegs
 ConfigRegs
 ConfigRegs
 ConfigRegs
 ConfigRegs
 ConfigRegs
 ConfigRegs
 ConfigRegs
 ConfigRegs
 AmigaDOS
 AmigaDOS
 AmigaDOS

PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 TYPE
 TYPE
 CONST
 CONST
 CONST
 CONST
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 CONST
 CONST
 CONST
 CONST
 TYPE
 TYPE
 PROCEDURE
 CONST
 PROCEDURE
 CONST
 CONST
 CONST
 CONST
 TYPE
 VAR
 CONST
 CONST
 CONST
 TYPE
 CONST
 PROCEDURE
 PROCEDURE
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 PROCEDURE
 TYPE
 TYPE

Definition Modules Cross Reference

Db1PF	Display	CONST
DBuffer	Rasters	CONST
DBufPacket	Gels	TYPE
DBufPacketPtr	Gels	TYPE
DC1	ASCII	CONST
DC2	ASCII	CONST
DC3	ASCII	CONST
DC4	ASCII	CONST
DeadEndAlert	Intuition	CONST
DeadPrefix	KeyMapResource	TYPE
DeadPrefixSet	KeyMapResource	TYPE
DEALLOCATE	Heap	PROCEDURE
Deallocate	Memory	PROCEDURE
DEALLOCATE	Storage	PROCEDURE
DEBlksPerTrack	FileHandler	CONST
Debug	Exec	PROCEDURE
Decimal	RealConversions	CONST
DecimalChars	ASCII	CONST
DeferRefresh	Intuition	CONST
DefFreq	NarratorDevice	CONST
DefMode	NarratorDevice	CONST
DefPitch	NarratorDevice	CONST
DefRate	NarratorDevice	CONST
DefSex	NarratorDevice	CONST
DefVol	NarratorDevice	CONST
DegToRad	MathLib0	PROCEDURE
DEInterleave	FileHandler	CONST
DEL	ASCII	CONST
Delay	AmigaDOSProcess	PROCEDURE
Delete	FileSystem	PROCEDURE
DeleteExtIO	IODevicesUtil	PROCEDURE
DeleteFile	AmigaDOS	PROCEDURE
DeleteLayer	Layers	PROCEDURE
DeletePort	PortsUtil	PROCEDURE
DeleteSet	LargeSets	PROCEDURE
DeleteStdIO	IODevicesUtil	PROCEDURE
DeleteSubString	Strings	PROCEDURE
DeleteTask	TasksUtil	PROCEDURE
DELowCyl	FileHandler	CONST
DeltaMove	Intuition	CONST
DEMemBufType	FileHandler	CONST
DENumBuffers	FileHandler	CONST
DENumHeads	FileHandler	CONST
DEPrefac	FileHandler	CONST
DEReservedBlks	FileHandler	CONST
DESecOrg	FileHandler	CONST
DESecsPerBlk	FileHandler	CONST
Designed	Text	CONST
DESizeBlock	FileHandler	CONST
Dest	Blit	CONST
DETableSize	FileHandler	CONST
DEUpperCyl	FileHandler	CONST

Modula-2 Software Construction Set Reference Guide

DevAbortIO	IODevices	CONST
DevBeginIO	IODevices	CONST
Device	IODevices	TYPE
DeviceData	PrinterBase	TYPE
DeviceDataPtr	PrinterBase	TYPE
DeviceList	AmigaDOSExt	TYPE
DeviceListPtr	AmigaDOSExt	TYPE
DeviceNode	FileHandler	TYPE
DeviceNodePtr	FileHandler	TYPE
DeviceProc	AmigaDOSProcess	PROCEDURE
DevicePtr	IODevices	TYPE
DFHID	DiskFont	CONST
DFtchMask	Display	CONST
Diab630	Preferences	CONST
DiabAdvD25	Preferences	CONST
DiabC150	Preferences	CONST
DiagArea	ConfigRegs	TYPE
DiagAreaPtr	ConfigRegs	TYPE
Diff	LargeSets	PROCEDURE
Disable	Interrupts	PROCEDURE
DiscResource	DiskResource	TYPE
DiscResourceFlags	DiskResource	TYPE
DiscResourceFlagsSet	DiskResource	TYPE
DiscResourcePtr	DiskResource	TYPE
DiscResourceUnit	DiskResource	TYPE
DiscResourceUnitPtr	DiskResource	TYPE
DiskFont	Text	CONST
DiskFontBase	DiskFont	VAR
DiskFontHeader	DiskFont	TYPE
DiskFontHeaderPtr	DiskFont	TYPE
DiskfontName	DiskFont	CONST
DiskInserted	Intuition	CONST
DiskName	DiskResource	CONST
DiskObject	Workbench	TYPE
DiskObjectPtr	Workbench	TYPE
DiskObjectType	Workbench	TYPE
DiskRemoved	Intuition	CONST
DisownBlitter	Blit	PROCEDURE
DisplayAlert	Intuition	PROCEDURE
DisplayBeep	Intuition	PROCEDURE
DisposeLayerInfo	Layers	PROCEDURE
DisposeRegion	Regions	PROCEDURE
DIVS32	System	PROCEDURE
DIVU32	System	PROCEDURE
DIWHorizPos	Display	CONST
DIWVrtc1Pos	Display	CONST
DIWVrtc1PosShift	Display	CONST
DLE	ASCII	CONST
DLTDevice	AmigaDOSExt	CONST
DLTDirectory	AmigaDOSExt	CONST
DLTVolume	AmigaDOSExt	CONST
DMAA11	DMAHardware	CONST

Definition Modules Cross Reference

DMAAud0	DMAHardware	CONST
DMAAud1	DMAHardware	CONST
DMAAud2	DMAHardware	CONST
DMAAud3	DMAHardware	CONST
DMAAudio	DMAHardware	CONST
DMABlitHog	DMAHardware	CONST
DMABlitter	DMAHardware	CONST
DMABltDone	DMAHardware	CONST
DMABltNZero	DMAHardware	CONST
DMAcopper	DMAHardware	CONST
DMADisk	DMAHardware	CONST
DMA Master	DMAHardware	CONST
DMARaster	DMAHardware	CONST
DMASetClr	DMAHardware	CONST
DMASprite	DMAHardware	CONST
DoCollision	Gels	PROCEDURE
DoIO	IODevices	PROCEDURE
done	FileSystem	CONST
Done	InOut	VAR
Done	LongInOut	VAR
Done	RealInOut	VAR
Done	TermInOut	VAR
DOSBase	System	VAR
DosInfo	AmigaDOSExt	TYPE
DosInfoPtr	AmigaDOSExt	TYPE
DosLibrary	AmigaDOSExt	TYPE
DosLibraryPtr	AmigaDOSExt	TYPE
DOSName	AmigaDOS	CONST
DosPacket	AmigaDOSProcess	TYPE
DosPacketPtr	AmigaDOSProcess	TYPE
DoubleClick	Intuition	PROCEDURE
DP1	KeyMapResource	CONST
DP2	KeyMapResource	CONST
DP2DFacShift	KeyMapResource	CONST
DP2DIndexMask	KeyMapResource	CONST
DPDead	KeyMapResource	CONST
DPMOD	KeyMapResource	CONST
DRActive	DiskResource	CONST
Draft	Preferences	CONST
DRA1loc0	DiskResource	CONST
DRA1loc1	DiskResource	CONST
DRA1loc2	DiskResource	CONST
DRA1loc3	DiskResource	CONST
DRA1locUnit	DiskResource	CONST
Draw	Drawing	PROCEDURE
DrawBorder	Intuition	PROCEDURE
DrawCircle	Drawing	PROCEDURE
DrawEllipse	Drawing	PROCEDURE
DrawerData	Workbench	TYPE
DrawerDataFileSize	Workbench	CONST
DrawerDataPtr	Workbench	TYPE
DrawGList	Gels	PROCEDURE

Modula-2 Software Construction Set Reference Guide

DrawImage	Intuition	PROCEDURE
DrawModeSet	Rasters	TYPE
DRF4	DiskResource	CONST
DRF5	DiskResource	CONST
DRF6	DiskResource	CONST
DRFreeUnit	DiskResource	CONST
DRGetUnit	DiskResource	CONST
DRGetUnitID	DiskResource	CONST
DRGiveUnit	DiskResource	CONST
Drive35	TrackDiskDevice	CONST
Drive525	TrackDiskDevice	CONST
DRLastComm	DiskResource	CONST
DRT37422D2S	DiskResource	CONST
DRTAmiga	DiskResource	CONST
DRTEEmpty	DiskResource	CONST
DskDMAOff	DiskResource	CONST
DSRCPR	ConsoleDevice	CONST
DualPF	Views	CONST
DupLock	AmigaDOS	PROCEDURE
e	MathLib0	CONST
Echo	InOut	VAR
Echo	TermInOut	VAR
ECIInt2Pend	ConfigRegs	CONST
ECIInt6Pend	ConfigRegs	CONST
ECIInt7Pend	ConfigRegs	CONST
ECIIntEna	ConfigRegs	CONST
ECIInterrupting	ConfigRegs	CONST
ECIReset	ConfigRegs	CONST
EExpansionBase	ConfigRegs	CONST
EExpansionSize	ConfigRegs	CONST
EExpansionSlots	ConfigRegs	CONST
EightLPI	Preferences	CONST
Elite	Preferences	CONST
EM	ASCII	CONST
EMemoryBase	ConfigRegs	CONST
EMemorySize	ConfigRegs	CONST
EMemorySlots	ConfigRegs	CONST
Enable	Interrupts	PROCEDURE
EndGadget	Intuition	CONST
EndRefresh	Intuition	PROCEDURE
EndRequest	Intuition	PROCEDURE
EndUpdate	Layers	PROCEDURE
ENQ	ASCII	CONST
Enqueue	Lists	PROCEDURE
entier	MathLib0	PROCEDURE
eof	Terminal	VAR
EOL	InOut	CONST
EOL	TermInOut	CONST
EOT	ASCII	CONST
Epson	Preferences	CONST
EpsonJX80	Preferences	CONST
equal	Strings	CONST

Definition Modules Cross Reference

ERFMemSpace	ConfigRegs	CONST
ERFNoShutup	ConfigRegs	CONST
ErrorActionNotKnown	AmigaDOS	CONST
ErrorBadStreamName	AmigaDOS	CONST
ErrorCommentTooBit	AmigaDOS	CONST
ErrorDeleteProtected	AmigaDOS	CONST
ErrorDeviceNotMounted	AmigaDOS	CONST
ErrorDirectoryNotEmpty	AmigaDOS	CONST
ErrorDirNotFound	AmigaDOS	CONST
ErrorDiskFull	AmigaDOS	CONST
ErrorDiskNotValidated	AmigaDOS	CONST
ErrorDiskWriteProtected	AmigaDOS	CONST
ErrorFileNotObject	AmigaDOS	CONST
ErrorInvalidComponentName	AmigaDOS	CONST
ErrorInvalidLock	AmigaDOS	CONST
ErrorInvalidResidentLibrary	AmigaDOS	CONST
ErrorLineTooLong	AmigaDOS	CONST
ErrorNoDefaultDir	AmigaDOS	CONST
ErrorNoDisk	AmigaDOS	CONST
ErrorNoFreeStore	AmigaDOS	CONST
ErrorNoMoreEntries	AmigaDOS	CONST
ErrorNotADosDisk	AmigaDOS	CONST
ErrorObjectExists	AmigaDOS	CONST
ErrorObjectInUse	AmigaDOS	CONST
ErrorObjectNotFound	AmigaDOS	CONST
ErrorObjectTooLarge	AmigaDOS	CONST
ErrorObjectWrongType	AmigaDOS	CONST
ErrorReadProtected	AmigaDOS	CONST
ErrorRenameAcrossDevices	AmigaDOS	CONST
ErrorSeekError	AmigaDOS	CONST
ErrorTaskTableFull	AmigaDOS	CONST
ErrorTooManyLevels	AmigaDOS	CONST
ErrorWriteProtected	AmigaDOS	CONST
ERTChainedConfig	ConfigRegs	CONST
ERTDiagValid	ConfigRegs	CONST
ERTMemBit	ConfigRegs	CONST
ERTMemList	ConfigRegs	CONST
ERTMemMask	ConfigRegs	CONST
ERTMemSize	ConfigRegs	CONST
ERTNewBoard	ConfigRegs	CONST
ERTTypeBit	ConfigRegs	CONST
ERTTypeMask	ConfigRegs	CONST
ERTTypeSize	ConfigRegs	CONST
ESC	ASCII	CONST
ESlotMask	ConfigRegs	CONST
ESlotShift	ConfigRegs	CONST
ESlotSize	ConfigRegs	CONST
ETB	ASCII	CONST
ETDClear	TrackDiskDevice	CONST
ETDFormat	TrackDiskDevice	CONST
ETDMotor	TrackDiskDevice	CONST
ETDRawRead	TrackDiskDevice	CONST

Modula-2 Software Construction Set Reference Guide

ETDRawWrite	TrackDiskDevice	CONST
ETDRead	TrackDiskDevice	CONST
ETDSeek	TrackDiskDevice	CONST
ETDUpdate	TrackDiskDevice	CONST
ETDWrite	TrackDiskDevice	CONST
ETX	ASCII	CONST
Examine	AmigaDOS	PROCEDURE
ExcElement	LargeSets	PROCEDURE
ExcRange	LargeSets	PROCEDURE
ExclusiveLock	AmigaDOS	CONST
ExecBase	ExecBase	TYPE
ExecBase	System	VAR
ExecBasePtr	ExecBase	TYPE
ExecName	Exec	CONST
Execute	AmigaDOS	PROCEDURE
Exit	AmigaDOSProcess	PROCEDURE
ExNext	AmigaDOS	PROCEDURE
exp	MathLib0	PROCEDURE
ExpansionBase	Expansion	VAR
ExpansionControl	ConfigRegs	TYPE
ExpansionControlPtr	ConfigRegs	TYPE
ExpansionName	Expansion	CONST
ExpansionRom	ConfigRegs	TYPE
ExpansionRomPtr	ConfigRegs	TYPE
Extended	Text	CONST
ExtractSubString	Strings	PROCEDURE
ExtraHalfBrite	Views	CONST
FABSd	System	PROCEDURE
FABSS	System	PROCEDURE
FADDd	System	PROCEDURE
FADDs	System	PROCEDURE
FanFold	Preferences	CONST
FCHID	DiskFont	CONST
FCMPd	System	PROCEDURE
FCMPs	System	PROCEDURE
FDIVd	System	PROCEDURE
FDIVs	System	PROCEDURE
Female	NarratorDevice	CONST
FF	ASCII	CONST
FibArchive	AmigaDOS	CONST
FibDelete	AmigaDOS	CONST
FibExecute	AmigaDOS	CONST
FibRead	AmigaDOS	CONST
FibWrite	AmigaDOS	CONST
File	FileSystem	TYPE
FileHandle	AmigaDOS	TYPE
FileHandlePtr	AmigaDOSExt	TYPE
FileHandleRecord	AmigaDOSExt	TYPE
FileInfoBlock	AmigaDOS	TYPE
FileInfoBlockPtr	AmigaDOS	TYPE
FileLock	AmigaDOS	TYPE
FileLockPtr	AmigaDOSExt	TYPE

FileLockRecord	AmigaDOSExt	TYPE
FileNameSize	Preferences	CONST
FilePtr	FileSystem	TYPE
FileSysStartupMsg	FileHandler	TYPE
FileSysStartupMsgPtr	FileHandler	TYPE
FillCarryIn	Blit	CONST
FillOr	Blit	CONST
FillXor	Blit	CONST
FindConfigDev	Expansion	PROCEDURE
FindName	Lists	PROCEDURE
FindPort	Ports	PROCEDURE
FindResident	Resident	PROCEDURE
FindSemaphore	Semaphores	PROCEDURE
FindTask	Tasks	PROCEDURE
FindToolType	Workbench	PROCEDURE
Fine	Preferences	CONST
FLOATd	System	PROCEDURE
FLOATs	System	PROCEDURE
FLONG	System	PROCEDURE
Flood	Drawing	PROCEDURE
FlushCList	CList	PROCEDURE
FMULd	System	PROCEDURE
FMULs	System	PROCEDURE
FNEGd	System	PROCEDURE
FNEGs	System	PROCEDURE
FollowMouse	Intuition	CONST
FontContents	DiskFont	TYPE
FontContentsHeader	DiskFont	TYPE
FontContentsHeaderPtr	DiskFont	TYPE
FontContentsPtr	DiskFont	TYPE
FontFlags	Text	TYPE
FontFlagsSet	Text	TYPE
FontStyle	Text	TYPE
FontStyleSet	Text	TYPE
Forbid	Interrupts	PROCEDURE
FreeBoardMem	Expansion	PROCEDURE
FreeCList	CList	PROCEDURE
FreeColorMap	Views	PROCEDURE
FreeConfigDev	Expansion	PROCEDURE
FreeCopList	Copper	PROCEDURE
FreeCprList	Copper	PROCEDURE
FreeDiskObject	Workbench	PROCEDURE
FreeEntry	Memory	PROCEDURE
FreeExpansionMem	Expansion	PROCEDURE
FreeFreeList	Workbench	PROCEDURE
FreeGBuffers	Gels	PROCEDURE
FreeHeap	Heap	PROCEDURE
FreeHoriz	Intuition	CONST
FreeList	Workbench	TYPE
FreeListPtr	Workbench	TYPE
FreeMem	Memory	PROCEDURE
FreePotBits	PotgoResource	PROCEDURE

Modula-2 Software Construction Set Reference Guide

FreeRaster	Rasters	PROCEDURE
FreeRemember	Intuition	PROCEDURE
FreeSignal	Tasks	PROCEDURE
FreeSprite	Sprites	PROCEDURE
FreeSysRequest	Intuition	PROCEDURE
FreeTrap	Tasks	PROCEDURE
FreeVert	Intuition	CONST
FreeVPortCopLists	Views	PROCEDURE
FREMD	System	PROCEDURE
FREMs	System	PROCEDURE
FrstDot	Rasters	CONST
FS	ASCII	CONST
FSHORT	System	PROCEDURE
FSUBd	System	PROCEDURE
FSUBs	System	PROCEDURE
GadgBackFill	Workbench	CONST
GadgDisabled	Intuition	CONST
Gadget	Intuition	TYPE
Gadget0002	Intuition	CONST
GadgetActivation	Intuition	TYPE
GadgetActivationSet	Intuition	TYPE
GadgetDown	Intuition	CONST
GadgetFlags	Intuition	TYPE
GadgetFlagsSet	Intuition	TYPE
GadgetMutualExcludeSet	Intuition	TYPE
GadgetPtr	Intuition	TYPE
GadgetType	Intuition	CONST
GadgetTypeSet	Intuition	TYPE
GadgetUp	Intuition	CONST
GadgHBox	Intuition	CONST
GadgHComp	Intuition	CONST
GadgHighBits	Intuition	CONST
GadgHImage	Intuition	CONST
GadgHNone	Intuition	CONST
GadgImage	Intuition	CONST
GadgImmediate	Intuition	CONST
GamePortTrigger	GamePortDevice	TYPE
GamePortTriggerPtr	GamePortDevice	TYPE
GetColProc	Gels	TYPE
GetGone	Gels	CONST
GelsInfo	Gels	TYPE
GelsInfoPtr	Gels	TYPE
GENLOC	GraphicsBase	CONST
GenLockAudio	Views	CONST
GenLockVideo	Views	CONST
GetCC	Exec	PROCEDURE
GetCLBuf	CList	PROCEDURE
GetCLChar	CList	PROCEDURE
GetCLWord	CList	PROCEDURE
GetColorMap	Views	PROCEDURE
GetCurrentBinding	Expansion	PROCEDURE
GetDefPrefs	Preferences	PROCEDURE

GetDiskObject	Workbench	PROCEDURE
GetGBuffers	Gels	PROCEDURE
GetIcon	Workbench	PROCEDURE
GetMsg	Ports	PROCEDURE
GetPacket	AmigaDOSProcess	PROCEDURE
GetPos	FileSystem	PROCEDURE
GetPrefs	Preferences	PROCEDURE
GetRGB4	Views	PROCEDURE
GetScreenData	Intuition	PROCEDURE
GetSprite	Sprites	PROCEDURE
GF0	Intuition	CONST
GF1	Intuition	CONST
GfxBase	GraphicsBase	TYPE
GfxBase	System	VAR
GfxBasePtr	GraphicsBase	TYPE
GimmeZeroZero	Intuition	CONST
GPCTAbsJoystick	GamePortDevice	CONST
GPCTAllocated	GamePortDevice	CONST
GPCTMouse	GamePortDevice	CONST
GPCTNoController	GamePortDevice	CONST
GPCTRelJoystick	GamePortDevice	CONST
GPDAskCType	GamePortDevice	CONST
GPDAskTrigger	GamePortDevice	CONST
GPDErrSetCType	GamePortDevice	CONST
GPDReadEvent	GamePortDevice	CONST
GPDSetCType	GamePortDevice	CONST
GPDSetTrigger	GamePortDevice	CONST
GPT10	GamePortDevice	CONST
GPT11	GamePortDevice	CONST
GPT12	GamePortDevice	CONST
GPT13	GamePortDevice	CONST
GPT14	GamePortDevice	CONST
GPT15	GamePortDevice	CONST
GPT2	GamePortDevice	CONST
GPT3	GamePortDevice	CONST
GPT4	GamePortDevice	CONST
GPT5	GamePortDevice	CONST
GPT6	GamePortDevice	CONST
GPT7	GamePortDevice	CONST
GPT8	GamePortDevice	CONST
GPT9	GamePortDevice	CONST
GPTDownKeys	GamePortDevice	CONST
GPTKeys	GamePortDevice	CONST
GPTKeysSet	GamePortDevice	TYPE
GPTUpKeys	GamePortDevice	TYPE
GraphicsName	GamePortDevice	CONST
greater	Graphics	CONST
GRelBottom	Strings	CONST
GRelHeight	Intuition	CONST
GRelRight	Intuition	CONST
GRelWidth	Intuition	CONST
GS	Intuition	CONST
	ASCII	CONST

Modula-2 Software Construction Set Reference Guide

GzzGadget	Intuition	CONST
HALTX	System	PROCEDURE
HAM	Views	CONST
HexadecimalChars	ASCII	CONST
HighBox	Intuition	CONST
HighComp	Intuition	CONST
HighFlags	Intuition	CONST
HighImage	Intuition	CONST
HighItem	Intuition	CONST
HighNone	Intuition	CONST
Hires	Views	CONST
HoldNModify	Display	CONST
HPLaserJet	Preferences	CONST
HPLaserJetPlus	Preferences	CONST
HSizeBits	Blit	CONST
HSizeMask	Blit	CONST
HT	ASCII	CONST
IconBase	Workbench	VAR
IconName	Workbench	CONST
IDCMPFlags	Intuition	TYPE
IDCMPFlagsSet	Intuition	TYPE
IDDosDisk	AmigaDOS	CONST
IDKickStartDisk	AmigaDOS	CONST
IDNoDiskPresent	AmigaDOS	CONST
IDNotReallyDos	AmigaDOS	CONST
IDUnreadableDisk	AmigaDOS	CONST
IDValidated	AmigaDOS	CONST
IDValidating	AmigaDOS	CONST
IDWriteProtected	AmigaDOS	CONST
IEClass	InputEvents	TYPE
IEClass5	InputEvents	CONST
IEClassActiveWindow	InputEvents	CONST
IEClassCloseWindow	InputEvents	CONST
IEClassDiskInserted	InputEvents	CONST
IEClassDiskRemoved	InputEvents	CONST
IEClassEvent	InputEvents	CONST
IEClassGadgetDown	InputEvents	CONST
IEClassGadgetUp	InputEvents	CONST
IEClassInactiveWindow	InputEvents	CONST
IEClassMax	InputEvents	CONST
IEClassMenuList	InputEvents	CONST
IEClassNewPrefs	InputEvents	CONST
IEClassNull	InputEvents	CONST
IEClassPointerPos	InputEvents	CONST
IEClassRawKey	InputEvents	CONST
IEClassRawMouse	InputEvents	CONST
IEClassRefreshWindow	InputEvents	CONST
IEClassRequester	InputEvents	CONST
IEClassSizeWindow	InputEvents	CONST
IEClassTimer	InputEvents	CONST
IECodeASCIIIdel	InputEvents	CONST
IECodeASCIIIFirst	InputEvents	CONST

IECodeASCIILast	InputEvents	CONST
IECodeC0First	InputEvents	CONST
IECodeC0Last	InputEvents	CONST
IECodeC1First	InputEvents	CONST
IECodeC1Last	InputEvents	CONST
IECodeCommCodeFirst	InputEvents	CONST
IECodeCommCodeLast	InputEvents	CONST
IECodeKeyCodeFirst	InputEvents	CONST
IECodeKeyCodeLast	InputEvents	CONST
IECodeLatin1First	InputEvents	CONST
IECodeLatin1Last	InputEvents	CONST
IECodeLButton	InputEvents	CONST
IECodeMButton	InputEvents	CONST
IECodeNewActive	InputEvents	CONST
IECodeNoButton	InputEvents	CONST
IECodeRButton	InputEvents	CONST
IECodeReqClear	InputEvents	CONST
IECodeReqSet	InputEvents	CONST
IECodeUpPrefix	InputEvents	CONST
IEQualifier	InputEvents	TYPE
IEQualifierCapsLock	InputEvents	CONST
IEQualifierControl	InputEvents	CONST
IEQualifierInterrupt	InputEvents	CONST
IEQualifierLAlt	InputEvents	CONST
IEQualifierLCommand	InputEvents	CONST
IEQualifierLeftButton	InputEvents	CONST
IEQualifierLShift	InputEvents	CONST
IEQualifierMidButton	InputEvents	CONST
IEQualifierMultiBroadcast	InputEvents	CONST
IEQualifierNumericPad	InputEvents	CONST
IEQualifierRAlt	InputEvents	CONST
IEQualifierRButton	InputEvents	CONST
IEQualifierRCommand	InputEvents	CONST
IEQualifierRelativeMouse	InputEvents	CONST
IEQualifierRepeat	InputEvents	CONST
IEQualifierRShift	InputEvents	CONST
IEQualifierSet	InputEvents	TYPE
IF23	Intuition	CONST
IF24	Intuition	CONST
IF25	Intuition	CONST
IF26	Intuition	CONST
IF27	Intuition	CONST
IF28	Intuition	CONST
IF29	Intuition	CONST
IF30	Intuition	CONST
Image	Intuition	TYPE
ImageNegative	Preferences	CONST
ImagePositive	Preferences	CONST
ImagePtr	Intuition	TYPE
in	InOut	VAR
In	LargeSets	PROCEDURE
InactiveWindow	Intuition	CONST

Modula-2 Software Construction Set Reference Guide

InclElement
 InclRange
 IncrCLMark
 INDDAddHandler
 INDDRemHandler
 INDSetMPort
 INDSetMTrig
 INDSetMType
 INDSetPeriod
 INDSetThresh
 INDWriteEvent
 Info
 InfoData
 InfoDataPtr
 InitAnimation
 InitArea
 InitBitMap
 InitCLPool
 InitCode
 InitGels
 InitGMasks
 InitMasks
 InitRastPort
 InitRequester
 InitResident
 InitSemaphore
 InitStruct
 InitTmpRas
 InitView
 InitVPort
 Input
 InputEvent
 InputEventPtr
 InRequest
 Insert
 InsertSubString
 InstallClipRegion
 InstallErrorHandler
 INTAud0
 INTAud1
 INTAud2
 INTAud3
 INTBlit
 INTCoper
 INTDskBlk
 INTDskSync
 Interlace
 Interrupt
 InterruptPtr
 Intersection
 INTExter
 INTInten

LargeSets
 LargeSets
 CList
 InputDevice
 InputDevice
 InputDevice
 InputDevice
 InputDevice
 InputDevice
 InputDevice
 InputDevice
 AmigaDOS
 AmigaDOS
 AmigaDOS
 Gels
 Areas
 Graphics
 CList
 Resident
 Gels
 Gels
 Gels
 Rasters
 Intuition
 Resident
 Semaphores
 Memory
 Rasters
 Views
 Views
 AmigaDOS
 InputEvents
 InputEvents
 Intuition
 Lists
 Strings
 Layers
 RunTimeErrors
 INTHardware
 INTHardware
 INTHardware
 INTHardware
 INTHardware
 INTHardware
 INTHardware
 INTHardware
 INTHardware
 Display
 Interrupts
 Interrupts
 LargeSets
 INTHardware
 INTHardware

PROCEDURE
 PROCEDURE
 PROCEDURE
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 PROCEDURE
 TYPE
 TYPE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 TYPE
 TYPE
 CONST
 PROCEDURE
 PROCEDURE
 PROCEDURE
 PROCEDURE
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 CONST
 TYPE
 TYPE
 PROCEDURE
 CONST
 CONST

IntNMI	Interrupts	CONST
INTPorts	INTHardware	CONST
INTRBF	INTHardware	CONST
INTSetClr	INTHardware	CONST
INTSoftInt	INTHardware	CONST
INTTBE	INTHardware	CONST
IntuiMessage	Intuition	TYPE
IntuiMessagePtr	Intuition	TYPE
IntuiText	Intuition	TYPE
IntuiTextLength	Intuition	PROCEDURE
IntuiTextPtr	Intuition	TYPE
IntuiTicks	Intuition	CONST
Intuition	IntuitionBase	PROCEDURE
IntuitionBase	IntuitionBase	TYPE
IntuitionBase	System	VAR
IntuitionBasePtr	IntuitionBase	TYPE
IntuitionName	Intuition	CONST
IntVector	Interrupts	TYPE
IntVectorPtr	Interrupts	TYPE
INTVertB	INTHardware	CONST
InversVid	Rasters	CONST
IOAudio	AudioDevice	TYPE
IOAudioPtr	AudioDevice	TYPE
IOClipReq	ClipboardDevice	TYPE
IOClipReqPtr	ClipboardDevice	TYPE
IODRPreq	PrinterDevice	TYPE
IODRPreqPtr	PrinterDevice	TYPE
IoErr	AmigaDOS	PROCEDURE
IOErrAborted	IODevices	CONST
IOErrBadLength	IODevices	CONST
IOErrNoCmd	IODevices	CONST
IOErrOpenFail	IODevices	CONST
IOExtPar	ParallelDevice	TYPE
IOExtParPtr	ParallelDevice	TYPE
IOExtSer	SerialDevice	TYPE
IOExtSerPtr	SerialDevice	TYPE
IOExtTD	TrackDiskDevice	TYPE
IOExtTDPtr	TrackDiskDevice	TYPE
IOFlagsSet	IODevices	TYPE
IOParAbort	ParallelDevice	CONST
IOParActive	ParallelDevice	CONST
IOParQueued	ParallelDevice	CONST
IOPArray	ParallelDevice	TYPE
IOPArrayPtr	ParallelDevice	TYPE
IOPrtCmdReq	PrinterDevice	TYPE
IOPrtCmdReqPtr	PrinterDevice	TYPE
IOPT4	ParallelDevice	CONST
IOPT5	ParallelDevice	CONST
IOPT6	ParallelDevice	CONST
IOPT7	ParallelDevice	CONST
IOPTPaperOut	ParallelDevice	CONST
IOPTPBusy	ParallelDevice	CONST

Modula-2 Software Construction Set Reference Guide

IOPTPSel	ParallelDevice	CONST
IOPTRWDir	ParallelDevice	CONST
IOQuick	IODevices	CONST
IORequest	IODevices	TYPE
IORequestPtr	IODevices	TYPE
IOSerAbort	SerialDevice	CONST
IOSerActive	SerialDevice	CONST
IOSerBufRead	SerialDevice	CONST
IOSerQueued	SerialDevice	CONST
IOST10	SerialDevice	CONST
IOST11	SerialDevice	CONST
IOST12	SerialDevice	CONST
IOST13	SerialDevice	CONST
IOST14	SerialDevice	CONST
IOST15	SerialDevice	CONST
IOST5	SerialDevice	CONST
IOST6	SerialDevice	CONST
IOST7	SerialDevice	CONST
IOST8	SerialDevice	CONST
IOST9	SerialDevice	CONST
IOStdReq	IODevices	TYPE
IOStdReqPtr	IODevices	TYPE
IOSTOverRun	SerialDevice	CONST
IOSTReadBreak	SerialDevice	CONST
IOSTWroteBreak	SerialDevice	CONST
IOSTXOffRead	SerialDevice	CONST
IOSTXOffWrite	SerialDevice	CONST
IOTArray	SerialDevice	TYPE
IOTArrayPtr	SerialDevice	TYPE
IOTDIndexSync	TrackDiskDevice	CONST
IsDrawn	Intuition	CONST
IsGrtrX	Clipping	CONST
IsGrtrY	Clipping	CONST
IsInteractive	AmigaDOS	PROCEDURE
IsLessX	Clipping	CONST
IsLessY	Clipping	CONST
Italic	Text	CONST
ItemAddress	Intuition	PROCEDURE
ItemEnabled	Intuition	CONST
ITEMNUM	Intuition	PROCEDURE
ItemText	Intuition	CONST
Jam1	Rasters	CONST
Jam2	Rasters	CONST
KBDAddResetHandler	KeyboardDevice	CONST
KBDReadEvent	KeyboardDevice	CONST
KBDReadMatrix	KeyboardDevice	CONST
KBDRemResetHandler	KeyboardDevice	CONST
KBDResetHandlerDone	KeyboardDevice	CONST
KCAIt	KeyMapResource	CONST
KCControl	KeyMapResource	CONST
KCDead	KeyMapResource	CONST
KCDownUp	KeyMapResource	CONST

KCNop	KeyMapResource	CONST
KCNoQual	KeyMapResource	CONST
KCShift	KeyMapResource	CONST
KCString	KeyMapResource	CONST
KCVanilla	KeyMapResource	CONST
KeyCodeB	Intuition	CONST
KeyCodeM	Intuition	CONST
KeyCodeN	Intuition	CONST
KeyCodeQ	Intuition	CONST
KeyCodeV	Intuition	CONST
KeyCodeX	Intuition	CONST
KeyMap	KeyMapResource	TYPE
KeyMapCode	KeyMapResource	TYPE
KeyMapCodeSet	KeyMapResource	TYPE
KeyMapNode	KeyMapResource	TYPE
KeyMapNodePtr	KeyMapResource	TYPE
KeyMapPtr	KeyMapResource	TYPE
KeyMapResource	KeyMapResource	TYPE
KeyMapResourcePtr	KeyMapResource	TYPE
KnobHit	Intuition	CONST
KnobHMin	Intuition	CONST
KnobVMin	Intuition	CONST
Lace	Views	CONST
LaceWB	Preferences	CONST
LargeSet	LargeSets	TYPE
Layer	Clipping	TYPE
LayerBackdrop	Clipping	CONST
LayerClipRectsLost	Clipping	CONST
LayerInfo	Clipping	TYPE
LayerInfoFlags	Clipping	TYPE
LayerInfoFlagsSet	Clipping	TYPE
LayerInfoPtr	Clipping	TYPE
LayerPtr	Clipping	TYPE
LayerRefresh	Clipping	CONST
LayersBase	System	VAR
LayerSimple	Clipping	CONST
LayerSmart	Clipping	CONST
LayersName	Layers	CONST
LayerSuper	Clipping	CONST
LayerUpdating	Clipping	CONST
Left0	AudioDevice	CONST
Left1	AudioDevice	CONST
LeftBorder	Intuition	CONST
LeftHit	Gels	CONST
Length	FileSystem	CONST
less	Strings	PROCEDURE
Letter	Preferences	CONST
LF	ASCII	CONST
LibBase	Libraries	CONST
LibChanged	Libraries	CONST
LibClose	Libraries	CONST
LibDelExp	Libraries	CONST

Modula-2 Software Construction Set Reference Guide

LibExpunge	Libraries	CONST
LibExtFunc	Libraries	CONST
LibFlags	Libraries	TYPE
LibFlagsSet	Libraries	TYPE
LibNonStd	Libraries	CONST
LibOpen	Libraries	CONST
Library	Libraries	TYPE
LibraryPtr	Libraries	TYPE
LibReserved	Libraries	CONST
LibSumming	Libraries	CONST
LibSumUsed	Libraries	CONST
LibUserDef	Libraries	CONST
LibVectSize	Libraries	CONST
LIF10	Clipping	CONST
LIF11	Clipping	CONST
LIF12	Clipping	CONST
LIF13	Clipping	CONST
LIF14	Clipping	CONST
LIF15	Clipping	CONST
LIF3	Clipping	CONST
LIF5	Clipping	CONST
LIF9	Clipping	CONST
LineMode	Blit	CONST
List	Lists	TYPE
ListPtr	Lists	TYPE
ln	MathLib0	PROCEDURE
ln2	MathLib0	CONST
LoadRGB4	Views	PROCEDURE
LoadSeg	AmigaDOS	PROCEDURE
LoadView	Views	PROCEDURE
LocateChar	Strings	PROCEDURE
LocateSubString	Strings	PROCEDURE
Lock	AmigaDOS	PROCEDURE
LockIBase	IntuitionBase	PROCEDURE
LockLayer	Layers	PROCEDURE
LockLayerInfo	Layers	PROCEDURE
LockLayerRom	RomLayers	PROCEDURE
LockLayers	Layers	PROCEDURE
log	MathLib0	PROCEDURE
LonelyMessage	Intuition	CONST
LongInt	Intuition	CONST
Lookup	FileSystem	PROCEDURE
LowCheckWidth	Intuition	CONST
LowCommWidth	Intuition	CONST
Lower	ASCII	CONST
MakeDosNode	Expansion	PROCEDURE
MakeFunctions	Libraries	PROCEDURE
MakeLibrary	Libraries	PROCEDURE
MakeScreen	Intuition	PROCEDURE
MakeVPort	Views	PROCEDURE
Male	NarratorDevice	CONST
MarkCList	CList	PROCEDURE

Definition Modules Cross Reference

MASM
MatchToolValue
MathBase
MathTransBase
MathTransName
MAWM
MaxBody
MaxBytesPerRow
MaxFontName
MaxFontPath
MaxFreq
MaxPitch
MaxPot
MaxRate
MaxSignal
MaxTabs
MaxTrap
MaxVol
MemBlockMask
MemBlockSize
MemChip
MemChunk
MemChunkPtr
MemClear
MemEntry
MemEntryPtr
MemFailed
MemFast
MemHeader
MemHeaderPtr
MemLargest
MemList
MemListPtr
MemPublic
MemReqSet
Menu
MenuCancel
MenuDown
MenuEnabled
MenuFlags
MenuFlagsSet
MenuHot
MenuItem
MenuItemFlags
MenuItemFlagsSet
MenuItemMutualExcludeSet
MenuItemPtr
MenuNull
MENUNUM
MenuPick
MenuPtr
MenuState

[illegible][illegible]

Modula-2 Software Construction Set Reference Guide

MenuToggle	Intuition	CONST
MenuToggled	Intuition	CONST
MenuUp	Intuition	CONST
MenuVerify	Intuition	CONST
MenuWaiting	Intuition	CONST
Message	Ports	TYPE
MessagePtr	Ports	TYPE
MF1	Intuition	CONST
MF2	Intuition	CONST
MF3	Intuition	CONST
MF4	Intuition	CONST
MF5	Intuition	CONST
MF6	Intuition	CONST
MF7	Intuition	CONST
MiDrawn	Intuition	CONST
MIF10	Intuition	CONST
MIF11	Intuition	CONST
MIF15	Intuition	CONST
MIF5	Intuition	CONST
MIF6	Intuition	CONST
MIF7	Intuition	CONST
MIF9	Intuition	CONST
MinFreq	NarratorDevice	CONST
MinList	Lists	TYPE
MinListPtr	Lists	TYPE
MinNode	Nodes	TYPE
MinNodePtr	Nodes	TYPE
MinPitch	NarratorDevice	CONST
MinRate	NarratorDevice	CONST
MinTermFlagsSet	Blit	TYPE
MinVol	NarratorDevice	CONST
MiscName	MiscResource	CONST
MiscResource	MiscResource	TYPE
MiscResourcePtr	MiscResource	TYPE
MLNM	ConsoleDevice	CONST
Mode640	Display	CONST
ModeNewFile	AmigaDOS	CONST
ModeOldFile	AmigaDOS	CONST
ModeReadWrite	AmigaDOS	CONST
ModifyIDCMP	Intuition	PROCEDURE
ModifyProp	Intuition	PROCEDURE
MouseButtons	Intuition	CONST
MouseMove	Intuition	CONST
mouthrb	NarratorDevice	TYPE
mouthrbPtr	NarratorDevice	TYPE
Move	Drawing	PROCEDURE
MoveLayer	Layers	PROCEDURE
MoveLayerInFrontOf	Layers	PROCEDURE
MoveScreen	Intuition	PROCEDURE
MoveSprite	Sprites	PROCEDURE
MoveWindow	Intuition	PROCEDURE
MRAallocMiscResource	MiscResource	CONST

MRFreeMiscResource	MiscResource	CONST
MrgCop	Views	PROCEDURE
MRParallelBits	MiscResource	CONST
MRParallelPort	MiscResource	CONST
MRSerialBits	MiscResource	CONST
MRSerialPort	MiscResource	CONST
MsgPort	Ports	TYPE
MsgPortPtr	Ports	TYPE
MTypeCloseDown	Workbench	CONST
MTypeDiskChange	Workbench	CONST
MTypeIOProc	Workbench	CONST
MTypePStd	Workbench	CONST
MTypeTimer	Workbench	CONST
MTypeToolExit	Workbench	CONST
MULS32	System	PROCEDURE
MULU32	System	PROCEDURE
MustDraw	Gels	CONST
NABC	Blit	CONST
NABNC	Blit	CONST
NAK	ASCII	CONST
NANBC	Blit	CONST
NANBNC	Blit	CONST
narratorrb	NarratorDevice	TYPE
narratorrbPtr	NarratorDevice	TYPE
NaturalF0	NarratorDevice	CONST
NDCantAlloc	NarratorDevice	CONST
NDExpunged	NarratorDevice	CONST
NDFreqErr	NarratorDevice	CONST
NDMakeBad	NarratorDevice	CONST
NDModeErr	NarratorDevice	CONST
NDNoAudLib	NarratorDevice	CONST
NDNoMem	NarratorDevice	CONST
NDNoWrite	NarratorDevice	CONST
NDPhonErr	NarratorDevice	CONST
NDPitchErr	NarratorDevice	CONST
NDRateErr	NarratorDevice	CONST
NDSexErr	NarratorDevice	CONST
NDUnimpl	NarratorDevice	CONST
NDUnitErr	NarratorDevice	CONST
NDVolErr	NarratorDevice	CONST
NewLayerInfo	Layers	PROCEDURE
NewLayerInfoCalled	Clipping	CONST
NewList	Lists	PROCEDURE
NewModifyProp	Intuition	PROCEDURE
NewPrefs	Intuition	CONST
NewRegion	Regions	PROCEDURE
NewScreen	Intuition	TYPE
NewScreenPtr	Intuition	TYPE
NewSize	Intuition	CONST
NewWindow	Intuition	TYPE
NewWindowPtr	Intuition	TYPE
NoCareRefresh	Intuition	CONST

Modula-2 Software Construction Set Reference Guide

NoCrossFill	Rasters	CONST
Node	Nodes	TYPE
NodePtr	Nodes	TYPE
NoIconPosition	Workbench	CONST
NoisyReq	Intuition	CONST
NoItem	Intuition	CONST
nomemory	FileSystem	CONST
NoMenu	Intuition	CONST
NormalFontStyle	Text	CONST
NoSignals	Tasks	CONST
NoSub	Intuition	CONST
notdone	FileSystem	CONST
NoTraps	Tasks	CONST
NTDevice	Nodes	CONST
NTFont	Nodes	CONST
NTFreeMsg	Nodes	CONST
NTInterrupt	Nodes	CONST
NTLibrary	Nodes	CONST
NTMemory	Nodes	CONST
NTMessage	Nodes	CONST
NTMsgPort	Nodes	CONST
NTProcess	Nodes	CONST
NTractor	Preferences	CONST
NTReplyMsg	Nodes	CONST
NTResource	Nodes	CONST
NTSC	GraphicsBase	CONST
NTSemaphore	Nodes	CONST
NTSignalSem	Nodes	CONST
NTSoftInt	Nodes	CONST
NTTask	Nodes	CONST
NTUnknown	Nodes	CONST
NUL	ASCII	CONST
NumMRTypes	MiscResource	CONST
NumSecs	TrackDiskDevice	CONST
NumUnits	TrackDiskDevice	CONST
ObtainConfigBinding	Expansion	PROCEDURE
ObtainSemaphore	Semaphores	PROCEDURE
ObtainSemaphoreList	Semaphores	PROCEDURE
OctalChars	ASCII	CONST
Octant1	Blit	CONST
Octant2	Blit	CONST
Octant3	Blit	CONST
Octant4	Blit	CONST
Octant5	Blit	CONST
Octant6	Blit	CONST
Octant7	Blit	CONST
Octant8	Blit	CONST
OffDisplay	DMAHardware	PROCEDURE
OffGadget	Intuition	PROCEDURE
OffMenu	Intuition	PROCEDURE
OffsetBeginning	AmigaDOS	CONST
OffsetCurrent	AmigaDOS	CONST

Modula-2 Software Construction Set Reference Guide

ParFlagsSet	ParallelDevice	TYPE
ParRadBoogie	ParallelDevice	CONST
ParShared	ParallelDevice	CONST
ParStatus	ParallelDevice	TYPE
ParStatusSet	ParallelDevice	TYPE
PASignal	Ports	CONST
PASoftInt	Ports	CONST
PDCmdQuery	ParallelDevice	CONST
PDCmdSetParams	ParallelDevice	CONST
PDErrBadDimension	PrinterDevice	CONST
PDErrBufferMemory	PrinterDevice	CONST
PDErrCancel	PrinterDevice	CONST
PDErrDimensionOverflow	PrinterDevice	CONST
PDErrInternalMemory	PrinterDevice	CONST
PDErrInvertHam	PrinterDevice	CONST
PDErrNotGraphics	PrinterDevice	CONST
PeekCLMark	CList	PROCEDURE
Permit	Interrupts	PROCEDURE
PF0	ParallelDevice	CONST
PF2	ParallelDevice	CONST
PF2Pri	Display	CONST
PF4	ParallelDevice	CONST
PF6	ParallelDevice	CONST
PF7	ParallelDevice	CONST
PFAAction	Ports	CONST
PFAFineScroll	Display	CONST
PFBA	Views	CONST
PFBFineScrollShift	Display	CONST
PFFineScrollMask	Display	CONST
pi	MathLib0	CONST
Pica	Preferences	CONST
PIF10	Intuition	CONST
PIF11	Intuition	CONST
PIF12	Intuition	CONST
PIF13	Intuition	CONST
PIF14	Intuition	CONST
PIF15	Intuition	CONST
PIF4	Intuition	CONST
PIF5	Intuition	CONST
PIF6	Intuition	CONST
PIF7	Intuition	CONST
PIF9	Intuition	CONST
PlanePtr	Graphics	TYPE
PlnCntrMsk	Display	CONST
PlnCntrShft	Display	CONST
PMBASM	ConsoleDevUnit	CONST
PMBAWM	ConsoleDevUnit	CONST
Point	Graphics	TYPE
PointerSize	Preferences	CONST
PointPtr	Graphics	TYPE
PointRel	Intuition	CONST
PolyDraw	Drawing	PROCEDURE

Modula-2 Software Construction Set Reference Guide

QueuePacket
QumeLP20
RadToDeg
Random
RasInfo
RasInfoPtr
RastPort
RastPortFlags
RastPortFlagsSet
RastPortPtr
RawDoFmt
RawKey
RawKeyConvert
Read
Read
Read
Read
ReadAgain
ReadCard
ReadCard
ReadChar
ReadExpansionByte
ReadExpansionRom
ReadInt
ReadInt
ReadLongCard
ReadLongInt
ReadPixel
ReadReal
ReadString
ReadString
ReadWord
ReadWrd
real
RealChars
RealToStringFormat
RecoveryAlert
Rectangle
RectanglePtr
RectFill
RefreshBits
RefreshGadgets
RefreshGList
RefreshWindow
RefreshWindowFrame
Region
RegionPtr
RegionRectangle
RegionRectanglePtr
Relation
ReleaseConfigBinding
ReleaseSemaphore

```
AmigaDOSProcess
Preferences
MathLib0
RandomNumbers
Rasters
Rasters
Rasters
Rasters
Rasters
Rasters
Exec
Intuition
ConsoleDevice
AmigaDOS
InOut
Terminal
TermInOut
Terminal
InOut
TermInOut
FileSystem
Expansion
Expansion
InOut
TermInOut
LongInOut
LongInOut
Drawing
RealInOut
InOut
TermInOut
FileSystem
InOut
MathLib0
ASCII
RealConversions
Intuition
Graphics
Graphics
Drawing
Intuition
Intuition
Intuition
Intuition
Intuition
Regions
Regions
Regions
Regions
Strings
Expansion
Semaphores
```

PROCEDURE
CONST
PROCEDURE
PROCEDURE
TYPE
TYPE
TYPE
TYPE
TYPE
TYPE
PROCEDURE
CONST
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
PROCEDURE
CONST
TYPE
CONST
TYPE
TYPE
PROCEDURE
CONST
PROCEDURE
PROCEDURE
CONST
PROCEDURE
TYPE
TYPE
TYPE
TYPE
TYPE
PROCEDURE
PROCEDURE

ReleaseSemaphoreList	Semaphores	PROCEDURE
RelVerify	Intuition	CONST
RemakeDisplay	Intuition	PROCEDURE
RemBob	Gels	PROCEDURE
RemConfigDev	Expansion	PROCEDURE
RemDevice	IODevices	PROCEDURE
Remember	Intuition	TYPE
RememberPtr	Intuition	TYPE
RemFont	Text	PROCEDURE
RemHead	Lists	PROCEDURE
RemIBob	Gels	PROCEDURE
RemICRVector	CIAResource	PROCEDURE
RemIntServer	Interrupts	PROCEDURE
RemLibrary	Libraries	PROCEDURE
Remove	Lists	PROCEDURE
Removed	Text	CONST
RemoveErrorHandler	RunTimeErrors	PROCEDURE
RemoveGadget	Intuition	PROCEDURE
RemoveGList	Intuition	PROCEDURE
RemPort	Ports	PROCEDURE
RemResource	Resources	PROCEDURE
RemSemaphore	Semaphores	PROCEDURE
RemTail	Lists	PROCEDURE
RemTask	Tasks	PROCEDURE
RemVSprite	Gels	PROCEDURE
Rename	AmigaDOS	PROCEDURE
ReplyMsg	Ports	PROCEDURE
Reportmouse	Intuition	CONST
ReportMouse	Intuition	PROCEDURE
ReqActive	Intuition	CONST
ReqClear	Intuition	CONST
ReqGadget	Intuition	CONST
ReqOffWindow	Intuition	CONST
ReqSet	Intuition	CONST
Request	Intuition	PROCEDURE
Requester	Intuition	TYPE
RequesterFlags	Intuition	TYPE
RequesterFlagsSet	Intuition	TYPE
RequesterPtr	Intuition	TYPE
ReqVerify	Intuition	CONST
Resident	Resident	TYPE
ResidentPtr	Resident	TYPE
Resource	Resources	TYPE
ResourcePtr	Resources	TYPE
Response	FileSystem	TYPE
RethinkDisplay	Intuition	PROCEDURE
ReturnError	AmigaDOS	CONST
ReturnFail	AmigaDOS	CONST
ReturnOk	AmigaDOS	CONST
ReturnWarn	AmigaDOS	CONST
RevPath	Text	CONST
RF10	Intuition	CONST

Modula-2 Software Construction Set Reference Guide

RF11	Intuition	CONST
RF3	Intuition	CONST
RF4	Intuition	CONST
RF5	Intuition	CONST
RF6	Intuition	CONST
RF7	Intuition	CONST
RF8	Intuition	CONST
RF9	Intuition	CONST
Right0	AudioDevice	CONST
Right1	AudioDevice	CONST
RightBorder	Intuition	CONST
RightHit	Gels	CONST
RingTrigger	Gels	CONST
RMBTrap	Intuition	CONST
RoboticF0	NarratorDevice	CONST
RomFont	Text	CONST
RootNode	AmigaDOSExt	TYPE
RootNodePtr	AmigaDOSExt	TYPE
RPDDumpRPort	PrinterDevice	CONST
RPF10	Rasters	CONST
RPF11	Rasters	CONST
RPF12	Rasters	CONST
RPF13	Rasters	CONST
RPF14	Rasters	CONST
RPF15	Rasters	CONST
RPF4	Rasters	CONST
RPF6	Rasters	CONST
RPF7	Rasters	CONST
RPF8	Rasters	CONST
RPF9	Rasters	CONST
RS	ASCII	CONST
RTAutoInit	Resident	CONST
RTCMachWord	Resident	CONST
RTColdStart	Resident	CONST
RTFlagsSet	Resident	TYPE
SatisfyMsg	ClipboardDevice	TYPE
SatisfyMsgPtr	ClipboardDevice	TYPE
SaveBack	Gels	CONST
SaveBob	Gels	CONST
SavePreserve	Gels	CONST
SBuf1024	Preferences	CONST
SBuf16000	Preferences	CONST
SBuf2048	Preferences	CONST
SBuf4096	Preferences	CONST
SBuf512	Preferences	CONST
SBuf8000	Preferences	CONST
SBufSizeBits	Preferences	CONST
Scientific	RealConversions	CONST
Screen	Intuition	TYPE
ScreenBehind	Intuition	CONST
ScreenFlags	Intuition	TYPE
ScreenFlagsSet	Intuition	TYPE

ScreenPtr	Intuition	TYPE
ScreenQuiet	Intuition	CONST
ScreenToBack	Intuition	PROCEDURE
ScreenToFront	Intuition	PROCEDURE
ScreenType	Intuition	CONST
ScrGadget	Intuition	CONST
ScrollLayer	Layers	PROCEDURE
ScrollRaster	Rasters	PROCEDURE
ScrollVPort	Views	PROCEDURE
SDCmdBreak	SerialDevice	CONST
SDCmdQuery	SerialDevice	CONST
SDCmdSetParams	SerialDevice	CONST
SDownBack	Intuition	CONST
SDragging	Intuition	CONST
Seed	RandomNumbers	VAR
Seek	AmigaDOS	PROCEDURE
SEF10	SerialDevice	CONST
SEF11	SerialDevice	CONST
SEF12	SerialDevice	CONST
SEF13	SerialDevice	CONST
SEF14	SerialDevice	CONST
SEF15	SerialDevice	CONST
SEF16	SerialDevice	CONST
SEF17	SerialDevice	CONST
SEF18	SerialDevice	CONST
SEF19	SerialDevice	CONST
SEF2	SerialDevice	CONST
SEF20	SerialDevice	CONST
SEF21	SerialDevice	CONST
SEF22	SerialDevice	CONST
SEF23	SerialDevice	CONST
SEF24	SerialDevice	CONST
SEF25	SerialDevice	CONST
SEF26	SerialDevice	CONST
SEF27	SerialDevice	CONST
SEF28	SerialDevice	CONST
SEF29	SerialDevice	CONST
SEF3	SerialDevice	CONST
SEF30	SerialDevice	CONST
SEF31	SerialDevice	CONST
SEF4	SerialDevice	CONST
SEF5	SerialDevice	CONST
SEF6	SerialDevice	CONST
SEF7	SerialDevice	CONST
SEF8	SerialDevice	CONST
SEF9	SerialDevice	CONST
SelectDown	Intuition	CONST
Selected	Intuition	CONST
SelectUp	Intuition	CONST
Semaphore	Semaphores	TYPE
SemaphorePtr	Semaphores	TYPE
SemaphoreRequest	Semaphores	TYPE

Modula-2 Software Construction Set Reference Guide

SemaphoreRequestPtr	Semaphores	TYPE
SendIO	IODevices	PROCEDURE
Ser7Wire	SerialDevice	CONST
SerEOFMode	SerialDevice	CONST
SerErrBaudMismatch	SerialDevice	CONST
SerErrBufErr	SerialDevice	CONST
SerErrBufOverflow	SerialDevice	CONST
SerErrDetectedBreak	SerialDevice	CONST
SerErrDevBusy	SerialDevice	CONST
SerErrInitErr	SerialDevice	CONST
SerErrInvBaud	SerialDevice	CONST
SerErrInvParam	SerialDevice	CONST
SerErrLineErr	SerialDevice	CONST
SerErrNoCTS	SerialDevice	CONST
SerErrNoDSR	SerialDevice	CONST
SerErrNotOpen	SerialDevice	CONST
SerErrParityErr	SerialDevice	CONST
SerErrPortReset	SerialDevice	CONST
SerErrTimerErr	SerialDevice	CONST
SerExtFlags	SerialDevice	TYPE
SerExtFlagsSet	SerialDevice	TYPE
SerFlags	SerialDevice	TYPE
SerFlagsSet	SerialDevice	TYPE
SerialName	SerialDevice	CONST
SerialPrinter	Preferences	CONST
SerParityOdd	SerialDevice	CONST
SerParityOn	SerialDevice	CONST
SerQueuedBrk	SerialDevice	CONST
SerRadBoogie	SerialDevice	CONST
SerShared	SerialDevice	CONST
SerStatus	SerialDevice	TYPE
SerStatusSet	SerialDevice	TYPE
SerXDisabled	SerialDevice	CONST
SetAfPt	Drawing	PROCEDURE
SetAPen	Drawing	PROCEDURE
SetBPen	Drawing	PROCEDURE
SetCollision	Gels	PROCEDURE
SetComment	AmigaDOS	PROCEDURE
SetCurrentBinding	Expansion	PROCEDURE
SetDMRequest	Intuition	PROCEDURE
SetDrMd	Drawing	PROCEDURE
SetDrPt	Drawing	PROCEDURE
SetExcept	Tasks	PROCEDURE
SetFont	Text	PROCEDURE
SetFunction	Libraries	PROCEDURE
SetICR	CIAResource	PROCEDURE
SetInfo	LargeSets	PROCEDURE
SetIntVector	Interrupts	PROCEDURE
SetMenuStrip	Intuition	PROCEDURE
SetOPen	Drawing	PROCEDURE
SetPointer	Intuition	PROCEDURE
SetPos	FileSystem	PROCEDURE

SetPrefs	Preferences	PROCEDURE
SetProtection	AmigaDOS	PROCEDURE
SetRange	LargeSets	PROCEDURE
SetRast	Rasters	PROCEDURE
SetRGB4	Views	PROCEDURE
SetRGB4CM	Views	PROCEDURE
SetSignal	Tasks	PROCEDURE
SetSoftStyle	Text	PROCEDURE
SetSR	Exec	PROCEDURE
SetTaskPri	Tasks	PROCEDURE
SetWindowTitles	Intuition	PROCEDURE
SetWrMsk	Drawing	PROCEDURE
SExtMark	SerialDevice	CONST
SExtMSPOn	SerialDevice	CONST
SF0	Intuition	CONST
SF1	Intuition	CONST
SF10	Intuition	CONST
SF10	PrinterDevice	CONST
SF11	Intuition	CONST
SF11	PrinterDevice	CONST
SF12	Intuition	CONST
SF12	PrinterDevice	CONST
SF13	Intuition	CONST
SF13	PrinterDevice	CONST
SF14	Intuition	CONST
SF14	PrinterDevice	CONST
SF15	Intuition	CONST
SF15	PrinterDevice	CONST
SF2	Intuition	CONST
SF3	Intuition	CONST
SF8	PrinterDevice	CONST
SF9	Intuition	CONST
SF9	PrinterDevice	CONST
SGRBBlack	ConsoleDevice	CONST
SGRBBlackBG	ConsoleDevice	CONST
SGRBlue	ConsoleDevice	CONST
SGRBlueBG	ConsoleDevice	CONST
SGRBold	ConsoleDevice	CONST
SGRC1r0	ConsoleDevice	CONST
SGRC1r0BG	ConsoleDevice	CONST
SGRC1r1	ConsoleDevice	CONST
SGRC1r1BG	ConsoleDevice	CONST
SGRC1r2	ConsoleDevice	CONST
SGRC1r2BG	ConsoleDevice	CONST
SGRC1r3	ConsoleDevice	CONST
SGRC1r3BG	ConsoleDevice	CONST
SGRC1r4	ConsoleDevice	CONST
SGRC1r4BG	ConsoleDevice	CONST
SGRC1r5	ConsoleDevice	CONST
SGRC1r5BG	ConsoleDevice	CONST
SGRC1r6	ConsoleDevice	CONST
SGRC1r6BG	ConsoleDevice	CONST

Modula-2 Software Construction Set Reference Guide

SGRC1r7	ConsoleDevice	CONST
SGRC1r7BG	ConsoleDevice	CONST
SGRCyan	ConsoleDevice	CONST
SGRCyanBG	ConsoleDevice	CONST
SGRDefault	ConsoleDevice	CONST
SGRDefaultBG	ConsoleDevice	CONST
SGRGreen	ConsoleDevice	CONST
SGRGreenBG	ConsoleDevice	CONST
SGRItalic	ConsoleDevice	CONST
SGRMagenta	ConsoleDevice	CONST
SGRMagentaBG	ConsoleDevice	CONST
SGRNegative	ConsoleDevice	CONST
SGRPrimary	ConsoleDevice	CONST
SGRRed	ConsoleDevice	CONST
SGRRedBG	ConsoleDevice	CONST
SGRUnderscore	ConsoleDevice	CONST
SGRWhite	ConsoleDevice	CONST
SGRWhiteBG	ConsoleDevice	CONST
SGRYellow	ConsoleDevice	CONST
SGRYellowBG	ConsoleDevice	CONST
ShadeBW	Preferences	CONST
ShadeColor	Preferences	CONST
ShadeGreyScale	Preferences	CONST
SharedLock	AmigaDOS	CONST
Shift	ASCII	CONST
ShortestForm	RealConversions	CONST
Showtitle	Intuition	CONST
ShowTitle	Intuition	PROCEDURE
SHShakeBits	Preferences	CONST
SHShakeNone	Preferences	CONST
SHShakeRTS	Preferences	CONST
SHShakeXON	Preferences	CONST
SI	ASCII	CONST
SigAbort	Tasks	CONST
SigBlit	Tasks	CONST
SigBreakCtrlC	AmigaDOS	CONST
SigBreakCtrlD	AmigaDOS	CONST
SigBreakCtrlE	AmigaDOS	CONST
SigBreakCtrlF	AmigaDOS	CONST
SigChild	Tasks	CONST
SigDos	Tasks	CONST
Signal	Tasks	PROCEDURE
SignalRange	Tasks	TYPE
SignalSemaphore	Semaphores	TYPE
SignalSemaphorePtr	Semaphores	TYPE
SignalSet	Tasks	TYPE
SignFlag	Blit	CONST
SigSingle	Tasks	CONST
SIHPriMask	Interrupts	CONST
SimpleRefresh	Intuition	CONST
SimpleSprite	Sprites	TYPE
SimpleSpritePtr	Sprites	TYPE

```

tr
t
r
ty1
ty2
ty3
ty4
tyMask
ols
ows
ols
ows
ls
ws
ed
et
etPtr

```

[illegible][illegible]

Modula-2 Software Construction Set Reference Guide

StdInput
StdOutput
StdScreenHeight
StrGadget
StringCenter
StringInfo
StringInfoPtr
StringLength
StringRight
STX
SUB
SubCList
SUBNUM
SubTime
SUD
SUL
SumKickData
SumLibrary
SuperBitMap
SuperState
SuperUnused
SUPFront
SUserFlags
SwapBitsRastPortClipRect
SWriteBits
SymmetricDiff
SYN
SyncSBitMap
SysGadget
SysRequest
T1
T2
T3
TallDot
tan
tanh
Task
TaskFlags
TaskFlagsSet
TaskPtr
TaskState
TaskStateSet
TBCHC1rTab
TBCHC1rTabsAll
TDAddChangeInt
TDAllowNon35
TDChangeNum
TDChangeState
TDErrBadDriveType
TDErrBadHdrSum
TDErrBadSecHdr
TDErrBadSecID

[illegible][illegible]

TDErrBadSecPreamble	TrackDiskDevice	CONST
TDErrBadSecSum	TrackDiskDevice	CONST
TDErrBadUnitNum	TrackDiskDevice	CONST
TDErrDiskChanged	TrackDiskDevice	CONST
TDErrDriveInUse	TrackDiskDevice	CONST
TDErrNoMem	TrackDiskDevice	CONST
TDErrNoSecHdr	TrackDiskDevice	CONST
TDErrNotSpecified	TrackDiskDevice	CONST
TDErrPostReset	TrackDiskDevice	CONST
TDErrSeekError	TrackDiskDevice	CONST
TDErrTooFewSecs	TrackDiskDevice	CONST
TDErrWriteProt	TrackDiskDevice	CONST
TDExtCom	TrackDiskDevice	CONST
TDFormat	TrackDiskDevice	CONST
TDGetDriveType	TrackDiskDevice	CONST
TDGetNumTracks	TrackDiskDevice	CONST
TDLabelSize	TrackDiskDevice	CONST
TDLastComm	TrackDiskDevice	CONST
TDMotor	TrackDiskDevice	CONST
TDName	TrackDiskDevice	CONST
TDProtStatus	TrackDiskDevice	CONST
TDRawRead	TrackDiskDevice	CONST
TDRawWrite	TrackDiskDevice	CONST
TDRemChangeInt	TrackDiskDevice	CONST
TDRemove	TrackDiskDevice	CONST
TDSecShift	TrackDiskDevice	CONST
TDSector	TrackDiskDevice	CONST
TDSeek	TrackDiskDevice	CONST
TDUPublicUnit	TrackDiskDevice	TYPE
TDUPublicUnitPtr	TrackDiskDevice	TYPE
termCH	InOut	VAR
termCH	TermInOut	VAR
TExcept	Tasks	CONST
Text	Text	PROCEDURE
TextAttr	Text	TYPE
TextAttrPtr	Text	TYPE
TextFont	Text	TYPE
TextFontPtr	Text	TYPE
TextLength	Text	PROCEDURE
TicksPerSecond	AmigaDOS	CONST
TimerBase	TimerDevice	VAR
timerequest	TimerDevice	TYPE
timerequestPtr	TimerDevice	TYPE
TimerName	TimerDevice	CONST
timeval	TimerDevice	TYPE
timevalPtr	TimerDevice	TYPE
TLaunch	Tasks	CONST
TmpRas	Rasters	TYPE
TmpRasPtr	Rasters	TYPE
ToggleSelect	Intuition	CONST
TopazEighty	Preferences	CONST
TopazSixty	Preferences	CONST

Modula-2 Software Construction Set Reference Guide

- TopBorder
- TopHit
- TProcTime
- TRAddRequest
- Translate
- TranslatorBase
- TranslatorName
- TrapRange
- TrapSet
- TRGetSysTime
- TRMakeBad
- TRNoMem
- TRNotUsed
- TRSetSysTime
- TRUNCd
- TRUNCs
- TSAdded
- TSExcept
- TSInvalid
- TSReady
- TSRemoved
- TSRun
- TStackChk
- TSWait
- TSwitch
- TypeOfMem
- UCopList
- UCopListPtr
- UCopperListInit
- Underlined
- UnGetCLChar
- UnGetCLWord
- Union
- Unit
- UnitActive
- UnitFlags
- UnitFlagsSet
- UnitInTask
- UnitMicroHz
- UnitPtr
- UnitVBlank
- UnLoadSeg
- UnLock
- UnlockIBase
- UnlockLayer
- UnlockLayerInfo
- UnlockLayerRom
- UnlockLayers
- UnPutCLChar
- UnPutCLWord
- UpfrontLayer
- Upper

Intuition
Gels
Tasks
TimerDevice
Translator
Translator
Translator
Tasks
Tasks
TimerDevice
Translator
Translator
Translator
TimerDevice
System
System
Tasks
Tasks
Tasks
Tasks
Tasks
Tasks
Tasks
Tasks
Tasks
Tasks
Memory
Copper
Copper
Copper
Text
CList
CList
LargeSets
IODevices
IODevices
IODevices
IODevices
IODevices
TimerDevice
IODevices
TimerDevice
AmigaDOS
AmigaDOS
IntuitionBase
Layers
Layers
RomLayers
Layers
CList
CList
Layers
ASCII

[illegible]

Modula-2 Software Construction Set Reference Guide

WBDevice	Workbench	CONST
WBDisk	Workbench	CONST
WBDiskMagic	Workbench	CONST
WBDiskVersion	Workbench	CONST
WBDrawer	Workbench	CONST
WBenchClose	Intuition	CONST
WBenchMessage	Intuition	CONST
WBenchMsg	System	VAR
WBenchOpen	Intuition	CONST
WBenchScreen	Intuition	CONST
WBenchToBack	Intuition	PROCEDURE
WBenchToFront	Intuition	PROCEDURE
WBenchWindow	Intuition	CONST
WBGarbage	Workbench	CONST
WBKick	Workbench	CONST
WBProject	Workbench	CONST
WBStartup	Workbench	TYPE
WBStartupPtr	Workbench	TYPE
WBTool	Workbench	CONST
WDownBack	Intuition	CONST
WDragging	Intuition	CONST
WF18	Intuition	CONST
WF19	Intuition	CONST
WF20	Intuition	CONST
WF21	Intuition	CONST
WF22	Intuition	CONST
WF23	Intuition	CONST
WF27	Intuition	CONST
WF28	Intuition	CONST
WF29	Intuition	CONST
WF30	Intuition	CONST
WF31	Intuition	CONST
WF6	Intuition	CONST
WF7	Intuition	CONST
WhichLayer	Layers	PROCEDURE
WideDot	Text	CONST
Window	Intuition	TYPE
WindowActive	Intuition	CONST
WindowClose	Intuition	CONST
WindowDepth	Intuition	CONST
WindowDrag	Intuition	CONST
WindowFlags	Intuition	TYPE
WindowFlagsSet	Intuition	TYPE
WindowLimits	Intuition	PROCEDURE
WindowPtr	Intuition	TYPE
WindowRefresh	Intuition	CONST
WindowSizing	Intuition	CONST
WindowTicked	Intuition	CONST
WindowToBack	Intuition	PROCEDURE
WindowToFront	Intuition	PROCEDURE
Write	AmigaDOS	PROCEDURE
Write	InOut	PROCEDURE

Definition Modules Cross Reference

Write	Terminal	PROCEDURE
Write	TermInOut	PROCEDURE
WriteCard	InOut	PROCEDURE
WriteCard	TermInOut	PROCEDURE
WriteChar	FileSystem	PROCEDURE
WriteExpansionByte	Expansion	PROCEDURE
WriteHex	InOut	PROCEDURE
WriteHex	TermInOut	PROCEDURE
WriteInt	InOut	PROCEDURE
WriteInt	TermInOut	PROCEDURE
WriteLn	InOut	PROCEDURE
WriteLn	Terminal	PROCEDURE
WriteLn	TermInOut	PROCEDURE
WriteLongCard	LongInOut	PROCEDURE
WriteLongHex	LongInOut	PROCEDURE
WriteLongInt	LongInOut	PROCEDURE
WriteLongOct	LongInOut	PROCEDURE
WriteOct	InOut	PROCEDURE
WriteOct	TermInOut	PROCEDURE
WritePixel	Drawing	PROCEDURE
WritePotgo	PotgoResource	PROCEDURE
WriteReal	RealInOut	PROCEDURE
WriteString	InOut	PROCEDURE
WriteString	Terminal	PROCEDURE
WriteString	TermInOut	PROCEDURE
WriteWord	FileSystem	PROCEDURE
WriteWrd	InOut	PROCEDURE
WTractor	Preferences	CONST
WUpFront	Intuition	CONST
XorRectRegion	Regions	PROCEDURE
XorRegionRegion	Regions	PROCEDURE

Appendix D Conversion from "C" to Modula-2.....	3
Comments.....	3
Identifiers.....	3
Integer Constants.....	3
Character Constants.....	4
Floating Point Constants.....	4
String Constants.....	4
lvalue.....	4
Simple Data Types.....	5
Pointer Type.....	6
Array Type.....	6
Structure Type.....	7
Union Type.....	7
Function Type.....	8
Boolean Type.....	8
Set Types.....	8
Type Conversion.....	8
Type Casting.....	9
"if" Statement.....	9
"while" Statement.....	9
"do" Statement.....	10
"for" Statement.....	10
"switch" Statement.....	10
"break" Statement.....	11
"continue" Statement.....	11
"return" Statement.....	11
"goto" Statement.....	12

Appendix D Conversion from 'C' to Modula-2.....	3
Comments.....	3
Identifiers.....	3
Integer Constants.....	3
Character Constants.....	4
Floating Point Constants.....	4
String Constants.....	4
Values.....	4
Single Data Types.....	5
Pointer Type.....	6
Array Type.....	6
Structure Type.....	7
Union Type.....	7
Function Type.....	8
Boolean Type.....	8
Set Types.....	8
Type Conversion.....	8
Type Casting.....	9
*if Statement.....	9
*while Statement.....	9
*do Statement.....	10
*for Statement.....	10
*switch Statement.....	10
*break Statement.....	11
*continue Statement.....	11
*return Statement.....	11
*goto Statement.....	12

The technical documentation available for the Amiga is oriented towards the language "C". Since it will be necessary to refer to that documentation it will be important to understand the way Modula-2 is related to "C".

The information presented here is designed to be a short guide to converting from "C" to Modula-2. Many of the language features available in "C" are available in Modula-2 in one form or another. Therefore an almost verbatim conversion of a "C" program is possible. To facilitate a conversion you must become familiar with some of the basic workings of "C" and the corresponding Modula-2 constructs.

Comments

In "C" comments begin with the sequence "/*" and end with "*/". For Modula-2 the same rules apply except the starting sequence is "(" and the ending sequence is "*)". Unlike in "C" comments in Modula-2 may be nested to any level.

Identifiers

"C" allows identifiers to be made up of a sequence of letters and digits and the underscore "_" character. Modula-2 defines identifiers as only letters and digits, however this implementation allows the underscore "_" character.

Integer Constants

An integer constant consists of a sequence of digits. Integer constants may be defined in several different bases: decimal, octal and hex. In "C" long integer constants are specified with the character "L" after the number in Modula-2 the same effect can be achieved using the character "D" after the number.

"C"	Modula-2	Description
10	10	decimal
0x4A	4AH	hexadecimal
0377	377B	octal
0xFC001000	0FC001000H	long hexadecimal
10000L	10000D	long decimal

Character Constants

A character constant in "C" is always defined as a character in single quotes. In Modula-2 a character may be defined in double or single quotes or as an octal value. In addition this implementation supports all of the "C" escape sequences which begin with a "\" character.

"C"	Modula-2	Description
'1'	'1'	normal character
'\n'	'\n'	newline character
'\10'	10C	octal character
'\44'	'\44'	octal character

Floating Point Constants

Floating point constants have the almost same representation in "C" as in Modula-2. The only minor difference is that Modula-2 requires at least one digit before the decimal point. This may require adding a zero digit before the decimal point if one does not already exist.

String Constants

"C" defines a string constant as characters between double quotes. In Modula-2 a string may be surrounded with single or double quotes. In addition this implementation supports all of the escape sequences which begin with "\". Refer to the chapter on the compiler for additional information. Both in "C" and Modula-2 strings are represented as a sequence of characters terminated by a null character.

lvalue

An lvalue is an expression referring to an addressable object. In Modula-2 this is called a designator and may consist of any number of pointer dereferences "^", array indexes "[]" or field references ".". The major difference between a designator and an lvalue is that in "C" an lvalue can be a full expression with operators, this is not allowed in Modula-2 directly.

"C"	Modula-2
*a = 1;	a^ := 1;
b->x := 2;	b^.x := 2;
c.y := 3;	c.y := 3;
a[3] := 4;	a[3] := 4;

Simple Data Types

Modula-2 has a richer set of data types than "C". Therefore every "C" type has a corresponding Modula-2 type, and in some cases it may be useful to convert a "C" type not to its direct equivalent but into a Modula-2 type which has a better mapping of the data being handled. The following table describes each type available in "C" and its direct equivalent in Modula-2. The "C" type "int" is not specifically listed because depending on the implementation it is equivalent to a "short" or a "long" "int". In the "Lattice C" implementation an "int" is equivalent to a "long". In the "Aztec C" implementation an "int" is equivalent to a "short".

Conversion from "C" Types to Modula-2 Types

"C"	Modula-2
char	CHAR
unsigned char	CHAR
short	INTEGER
unsigned short	CARDINAL
long	LONGINT
unsigned long	LONGCARD
float	REAL
double	REAL

Conversion from Amiga "C" Types to Modula-2 Types

"C"	Modula-2
BYTE	BYTE or CHAR
UBYTE	BYTE or CHAR
BYTEBITS	SET OF [0..7]
WORD	INTEGER
UWORD	CARDINAL
WORDBITS	SET OF [0..15] or BITSET
LONG	LONGINT
ULONG	LONGCARD
LONGBITS	SET OF [0..31]
STRPTR	ADDRESS or POINTER TO CHAR
APTR	ADDRESS
CPTR	ADDRESS
SHORT	INTEGER
USHORT	CARDINAL
FLOAT	REAL
DOUBLE	REAL
COUNT	INTEGER
UCOUNT	CARDINAL
BOOL	INTEGER or BOOLEAN
TEXT	CHAR

In the previous table the "C" types used by much of the Amiga software are shown with their Modula-2 equivalents. These types are defined in the "C" include file "exec/types.h".

Pointer Type

Pointers in "C" and Modula-2 are very similar and can be converted easily from one to the other.

Declaration:

"C"	Modula-2
struct Node *np;	np: POINTER TO Node;
char *cp;	cp: POINTER TO CHAR;

Usage:

"C"	Modula-2
np = &myNode;	np := ADR(myNode);
np->next = lastNode;	np^.next := lastNode;
*cp = '1';	cp^ := '1';

Array Type

Arrays in "C" are less flexible than in Modula-2 and therefore are easily converted to Modula-2. The main difference is in "C" an array always begins at index 0, while in Modula-2 the array index may begin at any value.

Declaration:

"C"	Modula-2
char msg[40]	msg: ARRAY [0..39] OF CHAR;
char map[10][10]	mtx: ARRAY [0..9],[0..9] OF CHAR

Usage:

"C"	Modula-2
msg[x] = '1';	msg[x] := '1';
map[x][y] = '*';	map[x][y] = '*';

Structure Type

The structure type in "C" is almost identical to the RECORD type in Modula-2. To convert from one to the other only the syntax is changed.

Declaration:

"C"

Modula-2

```
struct Node
```

```
{
  Node *next;
  long  x, y;
};
struct Node *np;
```

```
TYPE NodePtr = POINTER TO Node;
```

```
Node = RECORD
```

```
  next : NodePtr;
  x, y : LONGINT;
END;
```

```
VAR np : NodePtr;
```

Usage:

"C"

Modula-2

```
np->x = 10L;
```

```
np->y = 20L;
```

```
np^.x := 10D;
```

```
np^.y := 20D;
```

Union Type

The union type in "C" can be emulated using a variant RECORD in Modula-2. The form may look different but the actual result is identical. A union is simply a mapping of a given storage with several different types.

Declaration:

"C"

Modula-2

```
struct Node
```

```
{
  Node *next;
  union pt
  {
    long x;
    long y;
  };
};
struct Node *np;
```

```
TYPE NodePtr = POINTER TO Node;
```

```
Node = RECORD
```

```
  next : NodePtr;
  CASE :CARDINAL OF
    |0: x : LONGINT;
    |1: y : LONGINT;
  END;
END;
```

```
VAR np : NodePtr;
```

Usage:

"C"

Modula-2

```
np->x = 10L;
```

```
np->y = 20L;
```

```
np^.x := 10D;
```

```
np^.y := 20D;
```

Function Type

The function type in "C" is very similar to the procedure type in Modula-2. A variable declared as a procedure type is actually a pointer to a procedure which takes the specified parameters.

Boolean Type

"C" does not have an explicit BOOLEAN type instead integers are used as booleans. The values of TRUE and FALSE in both "C" and Modula-2 are the same. TRUE is equal to one, FALSE is equal to zero. The major difference is in "C" many times any non-zero value is considered as true, however in Modula-2 only the value specified by the constant TRUE is valid.

Set Types

In "C" set types are implemented using integers and "C" allows operations on integers such as ORing, ANDing, etc. In Modula-2 set types are a separate type specified as "SET OF ...". The following table gives the Modula-2 equivalents for typical bitwise related operations.

Table of Conversion for Set Operators

"C"	Modula-2	Description
	+	or
&	*	and
^	/	xor

Type Conversion

"C" performs implicit type conversion when operands of different types are used in an expression. Modula-2 does not perform implicit conversion, however the same effect can be achieved by using explicit coercion in Modula-2 also called type transfer.

Type Casting

Type casting is a facility available in "C" which is almost identical to type coercion in Modula-2. Type casting converts an operand of a given type into a different type. The difference between the two is the syntax used by the two languages.

"C"	Modula-2
(char)a	CHAR(a)
(struct *Node)tree	NodePtr(tree)
(long)x	LONGINT(x)

"if" Statement

The "if" statement in "C" is very similar to the "IF" statement in Modula-2. Except that in "C" any non-zero value is considered to be true, zero is always false. In Modula-2 the value must be of BOOLEAN type and be equal to either the constant TRUE or FALSE.

"C"	Modula-2
if (a == b)	IF a = b THEN
{	x := y;
x = y;	ELSE
}	y := x;
else	END;
{	
y = x;	
}	

"while" Statement

The "while" statement in "C" is very similar to the "WHILE" statement in Modula-2. Except that in "C" any non-zero value is considered to be true, zero is always false. In Modula-2 the value must be of BOOLEAN type and be equal to either the constant TRUE or FALSE.

"C"	Modula-2
while (a == b)	WHILE a = b DO
{	x := y;
x = y;	END;
}	

"do" Statement

The "do" statement in "C" is very similar to the "REPEAT" statement in Modula-2. Except that in "C" any non-zero value is considered to be true, zero is always false. In Modula-2 the value must be of BOOLEAN type and be equal to either the constant TRUE or FALSE. In addition the "do" statement ends on false, but the "REPEAT" statement ends on true.

"C"	Modula-2
do	REPEAT
{	x := y;
x = y;	UNTIL a <> b;
}	
while (a == b);	

"for" Statement

The "for" statement in "C" is similar to the "FOR" statement in Modula-2. The "for" statement in "C" allows any expression to be used for the initialization, limit and increment. Some times it is used in such a way that it can not be directly translated to Modula-2, in this situation a "WHILE" statement should be used to emulate the "for" statement.

"C"	Modula-2
for(x=1; x <= 10; x++)	FOR x := 1 TO 10 DO
{	y := x;
y = x;	END;
}	

"switch" Statement

The "switch" statement in "C" can be emulated in Modula-2 using the "CASE" statement. The "default" clause in a "switch" statement is equivalent to specifying the "ELSE" clause in a "CASE" statement.

"C"	Modula-2
switch (cmd)	CASE cmd OF
{	'P': PrinterLog();
case 'P': PrinterLog();	'L': ListFiles();
break;	ELSE
case 'L': ListFiles();	NoCmd();
break;	END;
default : NoCmd();	
break;	
}	

"break" Statement

There is no direct equivalent to the "break" statement available in Modula-2. One way to simulate the "break" statement is to use the "EXIT" statement within a "LOOP" statement. Statement sequences which use "break" may have to be changed to another form in order to work properly, this can be done easily in most cases.

"C"	Modula-2

for(;;)	LOOP
{	...
...	EXIT;
break;	...
...	END;
}	

"continue" Statement

There is no direct equivalent to the "continue" statement available in Modula-2. Statement sequences which use "continue" may have to be changed to another form in order to work properly, this can be done easily in most cases.

"return" Statement

The "return" statement in "C" is very similar to the "RETURN" statement in Modula-2. The main difference is that in a Modula-2 procedure which returns a value must have a "RETURN" statement to work properly, while in "C" the "return" statement is not required.

"C"	Modula-2

return(-1);	RETURN (-1);
return(sin(x+y));	RETURN sin(x+y);

"goto" Statement

The "goto" statement in "C" has no direct counterpart in Modula-2. However it is common practice to use "LOOP" and "EXIT" to simulate a "goto" statement rather elegantly.

"C"	Modula-2
for(;;)	LOOP
{	...
...	EXIT;
goto done;	...
...	EXIT;
goto done;	...
...	END;
}	
done:	

Appendix E Conversion From TDI To M2SCS.....	3
Compiler Differences.....	3
Standard Modules.....	5
Amiga Modules.....	6

Appendix E Conversion from TDI to MSCS	3
Compiler Differences	3
Standard Modules	3
Alias Modules	3

This appendix describes how a program written for TDI Modula-2 version 3.00 may be translated to work under the M2SCS environment. The differences between the two compilers is discussed as well as the differences in the libraries. The difficulty of conversion depends on the particular program, but generally is not very difficult, in most cases once the program is successfully compiled using M2SCS the program will have a very good chance of running the first time through.

Since the edit-compile cycle is very fast the changes which have to be made to the source can be performed and checked very quickly. The majority of the conversion process will probably involve changes relating to the library modules. However, M2SCS has almost every facility available in the TDI implementation so in most cases it will not be necessary to write additional modules when converting programs. Further, M2SCS in many cases gives you several choices for accomplishing the same function so the conversion is made much easier.

Compiler Differences

This section describes the differences between the compilers in the two implementation. Both compilers implement Modula-2 fully and so the differences are relatively minor. M2SCS provides some extensions not available in the TDI implementation but this will not effect the conversion of programs to M2SCS.

Both compilers implement compiler switches imbedded in user comments, however the switches have different meanings and it will be necessary to remove all compiler switches from source files before recompiling the source using M2SCS.

The TDI compiler implements sets larger than a machine word within the compiler, this is a very non-standard feature not defined by the language. M2SCS implements large sets but in a standard way, via a library module called "LargeSets". A large set is defined as a set which has more than 32 elements. To convert a program which uses this non-standard feature of the TDI compiler change operations on the set to procedure calls to the "LargeSets" module.

The TDI compiler implements the type ADDRESS as a POINTER TO WORD. M2SCS implements the type ADDRESS as a POINTER TO BYTE. The smallest addressable unit of storage on the MC68000 series is a BYTE not a WORD and so the POINTER TO BYTE definition is more accurate. Further, the TDI implementation uses ADDRESS to refer to objects which may actually not be on a WORD boundary, this does not cause any problems because in most cases the objects are not dereferenced using the ADDRESS type. In almost all cases no change needs to be performed in the source unless the ADDRESS is actually dereferenced. In the case the ADDRESS is dereferenced simply define a type called ADDRESSW as POINTER TO WORD and the same effect will be produced.

Constants in both compilers are handled very similar with one small exception, LONGINT/LONGCARD decimal constants have to be followed by the letter "D" in M2SCS. In the TDI implementation this is optional. In M2SCS an assignment of a CARDINAL/INTEGER to a LONGINT/LONGCARD can be performed without

any type coercion so that in many cases not specifying a "D" will not cause any problem however in an expression the "D" is not optional when one of the operands is of LONGINT/LONGCARD while the other is a decimal constant not followed by the character "D".

Since M2SCS is a single pass compiler a procedure has to be defined before it may be referenced. When translating a program it may be necessary to rearrange the procedures in the source file to avoid referencing procedures before they are defined. To allow procedures to be referenced before they are used M2SCS allows a procedure to be "FORWARD" defined, for more information on this feature refer to the compiler chapter of this manual.

The TDI compiler defines the standard constant NIL not as 0 but as -1, M2SCS defines NIL as 0. Almost all of the routines in the libraries which return pointers, return either a valid pointer or 0. So, TDI's choice of a value of -1 for NIL was not a very good one. However, the SYSTEM module in TDI's compiler defines a constant NULL which has a value of 0. When converting a module which has many uses of the constant NULL either perform a search and replace of NULL with NIL or at the beginning of the module define the constant NULL = NIL.

The TDI implementation allows a constant string to be passed to a VAR parameter specified as "ARRAY OF CHAR". This is not allowed in standard Modula-2. Within the procedure the address of this parameter is sometimes taken, the equivalent effect can be achieved in the following manner:

- * Declare parameter as ARRAY OF CHAR, non VAR
- * Use \$D compiler switch to disable copying of parameter

The following code fragments demonstrates the above stated ideas:

TDI Implementation:

```
PROCEDURE Node(VAR s: ARRAY OF CHAR);
VAR sp: ADDRESS;
BEGIN
  ...
  sp := ADR(s);
  ...
END Node;
```

M2SCS Implementation:

```
(* $D- disable copying of dynamic parameter *)
PROCEDURE Node(s: ARRAY OF CHAR);
VAR sp: ADDRESS;
BEGIN
  ...
  sp := ADR(s);
  ...
END Node;
(* $D+ enable copying of dynamic parameter *)
```

There may be other minor differences between the two compilers, but they should not pose any great difficulty to converting programs to M2SCS.

Standard Modules

This section describes the differences in the modules available in one form or another on most Modula-2 implementations.

The **FileSystem** module available in M2SCS is not available in the TDI implementation. A rough equivalent is the non-standard **Streams** module. The most important difference is that unlike the **Streams** module the **FileSystem** module buffers the data, this feature improves i/o performance by as much as 10 fold.

The **InOut** module is available in both the M2SCS and TDI implementations. The modules are nearly identical. If the **InOut** module will only be used for terminal i/o then it may be more efficient to use the **TermInOut** module. When using **InOut** for file i/o the M2SCS implementation is much more efficient because it relies on the **FileSystem** module which implements buffered i/o.

The **LongInOut** module is available in both the M2SCS and TDI implementations. The modules are nearly identical. When using **LongInOut** for file i/o the M2SCS implementation is much more efficient because it relies on the **InOut** module which in turn relies on the **FileSystem** module.

The **MathLib0** module is available in both the M2SCS and TDI implementations. The M2SCS module contains additional mathematical procedures not available in the TDI implementation. In M2SCS before using the procedures in the **MathLib0** module the Amiga's "mathtrans.library" must be opened successfully, this is simplified by using the **InitMathLib0** module.

The **RealInOut** module is available in both the M2SCS and TDI implementations. When using **RealInOut** for file i/o the M2SCS implementation is much more efficient because it relies on the **InOut** module which in turn relies on the **FileSystem** module.

The **Storage** module is available in both the M2SCS and TDI implementations. In the TDI implementation a specified amount of memory must be set aside for the heap. In the M2SCS implementation memory is only allocated from the system as it is actually requested by calls to **ALLOCATE**, in addition a variation of the **Storage** module is provided called **Heap**. The **Heap** module exports the same procedures as the **Storage** module however it provides another procedure called **FreeHeap** which frees all memory still in use, the **Heap** module is useful for programs which do not use **DEALLOCATE** to free memory.

The **Terminal** module is available in both the M2SCS and TDI implementations. The modules are nearly identical. The M2SCS **Terminal** module performs i/o through the **StdInput** and **StdOutput** file handles defined in the module **System**.

Amiga Modules

This section discusses the differences between the Amiga hardware/software related modules. To convert a program from TDI to M2SCS it may be necessary to import procedures, types, etc. from modules with different names than in the TDI implementation, a table at the end of this appendix show which module in the TDI implementation corresponds to a module in the M2SCS implementation.

In M2SCS ROM resident Amiga libraries are opened automatically by the startup code and closed by the exit code. Opening a ROM resident library does not cause any memory to be used and makes writing programs easier, disk resident libraries must still be explicitly opened by the user. In the TDI implementation libraries must be opened by the user's code explicitly. When convert a program remove the code which opens the ROM resident libraries. The following Amiga ROM resident libraries are opened and closed automatically by M2SCS:

- * exec.library (referenced directly at AbsExecBase)
- * dos.library
- * graphics.library
- * intuition.library
- * layers.library
- * mathffp.library

Some TDI programs exit by calling the AmigaDOS Exit procedure. When using M2SCS always exit by the module body of the main module or using the HALT standard procedure. The M2SCS exit code closes the libraries opened by the startup code and also handles the necessary communications with the WorkBench environment. Replace all calls to the Exit procedure with calls to the standard procedure HALT when converting from the TDI implementation.

In the TDI implementation many objects are passed to procedures via a pointer to the object, in M2SCS almost all parameters which pass fixed size structures are declared as VAR parameters. In order to compile successfully the pointers must be dereferenced using the '^' operator. A VAR parameter also causes the pointer to the object to be passed, the advantage to using VAR parameters is that type checking can not be disabled accidentally. Since ADDRESS is compatible with all pointer types, it would be easy to accidentally pass an ADDRESS for any pointer parameter without any warning from the compiler.

Procedures which may be passed different types of parameters are declared as ADDRESS parameters to allow a pointer to any type to be passed without type coercion. The other exception to using VAR parameters is when a NIL value may be passed for a procedure parameter.

The following example shows the conversion necessary for many of the procedures in the Amiga libraries.

TDI Declaration:

```
PROCEDURE CloseWindow(W: WindowPtr);
```

TDI Usage:

```
VAR
  wp: WindowPtr;
BEGIN
  ...
  CloseWindow(wp);
  ...
```

M2SCS Declaration:

```
PROCEDURE CloseWindow(VAR w: Window);
```

M2SCS Usage:

```
VAR
  wp: WindowPtr;
BEGIN
  ...
  CloseWindow(wp^);
  ...
```

The following table shows a rough correlation of TDI modules and their corresponding modules in M2SCS.

TDI Module	M2SCS Module(s)
ADKBits	ADKHardware
Alerts	Alerts
AmigaUtils	--
AMIGAX	System
ANSIConsole	--
ANSIPrinter	--
Areas	Areas
ASCII	ASCII
AudioDevice	AudioDevice
Blitter	Blit
BlitterHardware	Blit
CIAHardware	CIAHardware
CIAResource	CIAResource
ClipboardDevice	ClipboardDevice
CListLibrary	CList
Colors	Views
CommandLine	System

Modula-2 Software Construction Set Reference Guide

TDI Module

M2SCS Module(s)

ConfigRegs
 ConfigVars
 ConsoleDevice
 Conversions
 Copper
 CopperUtils
 CustomHardware
 Devices
 DiskFontLibrary
 DiskResource
 DMABits
 DMAUtils
 DOSCodeLoader
 DOSExtensions
 DOSFiles
 DOSLibrary
 DOSProcessHandler
 EasyGadgets
 Exec
 ExecBase
 Expansion
 FileHandler
 Gadgets
 GadgetUtils
 GamePortDevice
 Gels
 GraphicsBase
 GraphicsLibrary
 IconLibrary
 InOut
 InputDevice
 InputEvents
 IntBits
 Interrupts
 Intuition
 IntuiUtils
 IO
 KeyboardDevice
 Layers
 LayersLibrary
 Libraries
 Lists
 LongHyperMathLib
 LongInOut
 LongMathLib0
 M2Conversions
 MathLib0
 Memory

ConfigRegs
 Expansion
 ConsoleDevice
 Conversions, RealConversions
 Copper
 CopperUtil
 CustomHardware
 IODevices
 DiskFont
 DiskResource
 DMAHardware
 DMAHardware
 AmigaDOS
 AmigaDOSExt
 AmigaDOS
 AmigaDOS
 AmigaDOSProcess
 SimpleGadgets \$\$
 Exec
 ExecBase
 Expansion
 FileHandler
 Intuition
 SimpleGadgets \$\$
 GamePortDevice
 Gels
 GraphicsBase
 Graphics, Blit
 Workbench
 InOut, TermInOut
 InputDevice
 InputEvents
 INTHardware
 Interrupts
 Intuition
 Intuition
 IODevices
 KeyboardDevice
 RomLayers
 Clipping, Layers
 Libraries
 Lists
 --
 LongInOut
 --
 Conversions, RealConversions
 MathLib0
 Memory

TDI Module	M2SCS Module(s)
-----	-----
Menus	Intuition
MenuUtils	SimpleMenus \$\$
NarratorDevice	NarratorDevice
Nodes	Nodes
ParallelDevice	ParallelDevice
Pens	Drawing
PenUtils	Drawing
Ports	Ports
PortUtils	PortsUtil
PotgoResource	PotgoResource
Preferences	Preferences
PrinterDevice	PrinterDevice
PrtBase	PrinterBase
RandomNumbers	RandomNumbers
Rasters	Rasters
RealInOut	RealInOut
Regions	Regions
Requesters	Intuition
Resident	Resident
Resources	Resources
RoundRobinScheduler	--
Screens	Intuition
Semaphores	Semaphores
SerialDevice	SerialDevice
Sprites	Sprites
Storage	Storage, Heap
Streams	FileSystem
Strings	Strings
Tasks	Tasks
Terminal	Terminal
Text	Text
TimerDevice	TimerDevice
TrackDiskDevice	TrackDiskDevice
TranslatorLibrary	Translator
Trapper	RunTimeErrors
Views	Views
Windows	Intuition
Workbench	Workbench

Key:

-- = no equivalent module

\$\$ = module part of add-on product available separately

Appendix F Compatibility and Portability.....	3
Modula-2 Compiler.....	3
Standard Modules.....	3
Non Standard Modules.....	4

Appendix F: Compatibility and Portability.....	3
Module-2 Compiler.....	3
Standard Modules.....	3
Non Standard Modules.....	4

This appendix contains notes on compatibility between this Modula-2 implementation and other Modula-2 implementations. This information will be useful when moving programs to or from other systems. In addition there are some notes on writing portable Modula-2 programs.

A well written Modula-2 program can be easily moved to another system as long as the target system provides the library modules that are imported by the program. If the target system does not provide the necessary library modules the program can still be moved but the missing library modules will need to be implemented.

Modula-2 Compiler

The compiler included in this package conforms very closely to the Modula-2 standard as described in the book "Programming In Modula-2" by Niklaus Wirth. The only minor deviation is that a procedure identifier has to be declared before it is used. To call a procedure before it is declared the heading of the procedure is declared FORWARD. This deviation is present in many of the newer Modula-2 compilers because the advantages of a one pass compiler far outweighs the disadvantages of declaring a FORWARD procedure.

If portability of the program is very important the pseudo-module SYSTEM should be avoided when ever possible. The module contains explicitly system dependent features for this implementation. Among implementation of Modula-2 on a given machine architecture the module SYSTEM should be fairly similar. However, on very different architectures such as a MC68000 micro-processor and an IBM 370 mainframe computer the module SYSTEM may be very different.

The compiler contains many useful enhancements such as new types and "C" language style string escape sequences. These enhancements may be used freely, but keeping in mind that they may not be available in every implementation of Modula-2. The following is a list of some of the enhancements:

- * 32-bit types LONGINT, LONGCARD
- * System dependent types BYTE, WORD, LONGWORD
- * Identifiers may contain underscore characters "_"
- * String literals allow escape sequences
- * FORWARD procedure declarations
- * CODE procedure declarations

Standard Modules

This implementation as well as most implementations of Modula-2 provide the following basic set of library modules:

- * FileSystem
- * InOut
- * RealInOut
- * Storage
- * Terminal

Modula-2 Software Construction Set Reference Guide

A program which used only these basic modules could be easily moved to other systems with virtually no changes. Regrettably a program which used only the above mentioned modules would not be able to take advantage of many of the unique features of a given system. This is especially true of a feature rich system such as the Amiga. However, for some applications these modules may be sufficient to develop useful programs. For example a text formatter, a data base manager and many other applications could be easily written using this basic set of modules.

Non Standard Modules

In addition to the standard library modules supplied with this package a large number of non-standard library modules are provided. The non-standard modules fall into two general categories:

- * Amiga specific modules (ROM Kernel, Intuition, etc...)
- * Non Amiga specific modules (Strings, Files, etc...)

Amiga specific modules allow a Modula-2 program to access the hardware of the Amiga as well as the many useful resident libraries built into the Amiga. A program which uses these modules will not be directly portable to other systems.

If a program uses the various features of the Amiga such as windows, menus, etc. it may still be made portable by writing a layer of software between the core program and the Amiga specific modules. By introducing a system independent layer to the program it will be possible to port the program to system such as the Atari ST and the Apple Macintosh.

The non Amiga specific modules such as CMemory, CString, CStdIO and others will be provided in future products for other computer systems. Using these modules will allow portability to systems on which this product is available. However, Modula-2 implementations from other vendor may not contain support for these modules.

Standard Modules

- * FileSystem
- * InOut
- * RealInOut
- * Storage
- * Terminal

Appendix G	Demonstration Programs.....	3
------------	-----------------------------	---

This appendix describes briefly the demonstration programs included with this package. The demonstration programs are provided without extensive documentation or support. The demonstration programs have been tested but may contain bugs and/or poor error checking. The main purpose of the demonstration programs is to provide the user with a lot of sample code from which to learn how to program on the Amiga using Modula-2. The source code maybe freely used in any user created programs unless otherwise specified in the source. The demonstration programs range from the very complex to the trivial. Many of the programs not written by our company were found on bulletin boards and various communications networks, the programs were moved to the M2SCS environment so that users would have more examples of Modula-2 code. Demo programs not written by our company are essentially included on disk for the users convenience and are not to be considered as being sold by the company. The authors of some of the demonstration programs ask that you send them a small contribution for their effort, we encourage you to do this if you find that program useful in some way.

Programs which consist of more than one module are contained in separate directories on the source demo disks. Each directory contains a file called "make", this file contains the proper compilation order for the program or programs in that directory. The "make" file may be used to manually compile each file or by sending the file to the standard input of the compiler to have all of the files compiled automatically. This can be done by making the directory containing the source files the current directory and then issuing the following command at the CLI prompt:

```
M2 <make
```

After the source files have been successfully compiled they may be linked into executable files ready to be run.

The following is a list of the currently included demonstration programs. This list may not be completely up to date with the actual contents of the disks, new programs may have been added or older programs removed. A name which is followed by (Dir) indicates the name of a sub-directory containing the program.

Animal (Dir)

An animal guessing game. The user picks an animal and then the computer tries to guess the name of the animal. This is a text oriented program.

BigSets (Dir)

A library module which implementation sets which are larger than a machine word. A test program for this new library module is included. This is a text oriented program.

Case (Dir)

A Modula-2 case converter. This program reads a Modula-2 source file converting all key words to upper case and other identifiers to the case in which they were found first. This is a text oriented program.

Chaos (Dir)

A complex mathematically oriented graphics program which allows you to explore Henon Mapping. This program uses Intuition, IFF and many other features of the Amiga.

CtoM2 (Dir)

A very primitive "C" to Modula-2 translator. This program performs simple keyword substitutions. This is a text oriented program.

Doodle (Dir)

A very simple drawing program. This program uses Intuition extensively.

DUtil (Dir)

A very nice directory management program which can serve as a replacement for the less than friendly CLI user interface. This program allows many different operations to be performed on one or more files using the mouse. This program uses Intuition and AmigaDOS extensively.

Gels (Dir)

This directory contains examples of using the Graphics Elements(GELS) routines built into the ROM Kernel. The programs in this directory use the image resource utility "AddImg" to append imagery to the executable files.

GWars (Dir)

This is an excellent one or two player strategy game with very high quality graphics and sound effects. The object of the game is to destroy an opponents space ship by firing a missile. This would be easy except that the planets which occupy the screen have varying gravity fields which cause the missiles to act in ways which are difficult to predict. The imagery for the various planets used in the game is implemented using the image resource utility "AddImg". This program uses Intuition, graphics and floating point math extensively.

HP (Dir)

A nice calculator program which emulates some of the commands of the HP-10C calculator. This program uses Intuition and floating point math.

Kermit (Dir)

A tele-communications program which supports the kermit protocol. This program provides a good example of using the "serial.device". The major functions of this program seems to work reliably. The current implementation busy waits on input which is not very efficient but otherwise is a very good example.

Mondrian (Dir)

This program randomly generates modern art drawings. The parameters of the drawing can be controlled through menu selections. The colors of the custom screen can be adjusted with the program's color requester. The color requester is in a separate module and maybe pulled out and used in user graphics programs. This program uses Intuition and graphics functions.

Othello (Dir)

Play the game of Othello against the computer. This program plays a very good game of Othello and is not very easy to beat. The difficulty level of the game can be adjusted through a menu selection. The higher the difficulty the longer the computer will take to make its move.

ShowDevs (Dir)

A number of library modules which implement some AmigaDOS utility functions. A test program is included which partially emulates the AmigaDOS ASSIGN command.

Spiro (Dir)

A very good graphics program which draws various patterns on the screen. The program uses Intuition and graphics functions extensively.

Std (Dir)

This directory contains about 50 simple programs found in an archive on a bbs. All of the programs in this directory are text oriented.

TestMods (Dir)

This directory contains a collection of programs which were used to test some of the library modules included in this package. These programs also serve as examples of how to use the various modules.

Trails (Dir)

A very good interactive graphics program. This program uses Intuition and graphics functions extensively.

WarpText (Dir)

This directory contains library modules for performing very fast text output. These routines are written in assembly language for maximum speed. These routines are limited to output on byte boundaries and can not be used to output to a normal window. Programs to demonstrate how the modules are to be used are also included.

XRef (Dir)

A Modula-2 source text cross reference utility. This program produces a file containing the original source with a line number on each line, at the end of this file is a list of the symbols in the file with a list of the line numbers on which each symbol may be found. This program is text oriented.

XText (Dir)

This directory contains library modules for performing very fast text output to any window. These routines are written in assembly language for maximum speed. Unlike the WarpText routines these routines are not limited to output on byte boundaries and can therefore be used to output to any window. A program to demonstrate how to use the module are also included.

Misc (Dir)

This directory contains a large number of good demonstration programs. Each source file in this directory is a complete program. The following is a description of each of the programs in this directory.

AudioLab

Audio device experimentation laboratory. Draw a waveform using the mouse and then play the waveform.

AudioTools

Procedures for playing simple music by queueing up notes. The procedures in this module simplify the use of the audio device.

Bounce

A 3D wire frame cube which bounces towards and away from the screen. This demo uses double buffered animation.

Break

An example of how to detect the user typing Ctrl-c from the CLI to interrupt a program.

CAD

An example of a very simple Computer Aided Design (CAD) program.

Clstuff

This example contains code to determine the WindowPtr for a CLI window.

ConsoleDemo

An example of directly accessing the "console.device".

CopperDemo

An example of creating a user copper list. This copper list causes new values to be loaded into the color registers in the middle of the screen. This change effectively displays 64 colors simultaneously on the screen.

CopperIntDemo

An example of creating a user copper list. This copper list contains instructions to load the color registers as well as an instruction to generate a copper interrupt.

Cube

A very impressive animation of a 3D solid cube. This demo uses double buffered animation.

DBufDemo

Demonstration of double buffered drawing. The area functions are used to draw the graphics.

Draw

A fairly complete free hand drawing program. This program is a good example of how to use Intuition. The drawing program even has a magnification window for examining images very closely.

DualPlay

An example of dynamically adding an addition bitplane to the WorkBench screen. After adding the bitplane some lines are drawn into this bitplane.

FHTDemo

Demonstration of the Fast Hartly Transform mathematical algorithm. This is very similar to the FFT used in many engineering applications.

HAM

Example of the Hold-and-Modify(HAM) graphics mode of Amiga.

IFFCheck

Check an IFF file for consistency. The names of the chunks found in the given IFF file are displayed.

IntuiLab

Intuition laboratory for experimenting with Intuition functions without having to perform all of the necessary setup.

LargeSetsDemo

Example of using the LargeSets module included in this package.

LayersDemo

Example of using the "layers.library" built into the ROM Kernel.

Lines

Example of opening Intuition window and drawing some random lines into the window.

LinesDemo

Example of drawing lines very quickly to a screen.

Prime

The famous sieve performance benchmark.

QClock

This program displays the current time and the amount of memory free in the system. This data is displayed in the title bar of a window.

Queens

The solution to the eight queens problem. Each of the possible board positions is displayed graphically.

Modula-2 Software Construction Set Reference Guide

RayTrace

The very popular ray tracing graphics program. This program may be used to generate some breathtaking pictures using the HAM graphics mode.

Robot

Type text on the keyboard and the Amiga will convert the text to speech. While the speech is being played a robot mouth will move in synchronization with the speech.

SBM

An example of the Intuition Super BitMap window. Random lines are drawn into a virtual window. The contents of the window may be output to the printer.

ShowArgs

This program displays the arguments listed on the command line if any.

ShowILBM

A utility for displaying pictures stored in IFF ILBM format. This program maybe used from either the CLI or the WorkBench. Any number of filenames may be specified.

Sier

A graphics program which draws sierpinsky curves.

SimpleTest

A test program for some of the "Simple" modules.

Sparks

A graphics program with lines bouncing from the edges of the display.

SpeechDemo

A demonstration of using the Amiga's text to speech converter routines.

Stereo

An example of stereo sound output via the audio hardware.

Sweep

An example of using the audio hardware, the generated sound varies as the mouse is moved up and down.

TaskDemo

A demonstration of spawning multiple tasks. Each task draws rectangles of a certain color. After all the tasks have terminated the program exits.

TermInOutDemo

An example of using the TermInOut module included with this package.

TimerDemo

An example of using the "timer.device".

WBWindow

An example of how to safely close the default output window opened when programs are executed from the WorkBench.

WindowDemo

A simple demonstration of how to open a window and receive mouse movement events.

TimerDemo

An example of using the "timer.device".

WbWindow

An example of how to safely close the default output window opened when programs are executed from the Workbench.

WindowDemo

A simple demonstration of how to open a window and receive mouse movement events.

Appendix H Glossary.....3

This appendix contains a glossary of technical terms used through out this manual. Each term listed here has a brief explanation. For addition information refer to other technical documentation.

ASCII	An acronym for American Standard Code for Information Interchange. A set of character code between 0 and 127 representing printing and non-printing characters.
Assembler	A program which takes as input an assembly language source file. The output of this program is a binary representation of the file. An assembler which supports macros is called a macro assembler.
Assembly Language	A high-level representation of the native language of the computer. Each machine level instruction is represented as a symbolic command. Assembly language is useful for programing short but very efficient functions. The language is very difficult to learn and is uneconomical to use for large projects.
Binary	A number represented in base two. A binary number consists of ones and zeros, the number of ones and zeros determines the range of numbers which may be represented.
Buffered I/O	Input and/or output operations which are performed through a local buffer. Buffered i/o can increase dramatically the performance of an i/o intensive operation because it avoids the overhead of calling the operating system to output small pieces of data. Rather the data is stored in a buffer and when the buffer becomes full the data is written to the actual device.
Byte	A basic unit of storage in a computer. A byte contains eight bits of data.
Character	An alpha-numeric symbol represented by an 8-bit binary code.
CLI	The (C)ommand (L)ine (I)nterface provides a user interface to the operating system. Commands are available for manipulating files and running programs. Refer to the AmigaDOS reference manual for documentation on the CLI.
"C" Language	An older programming language with similarities to PASCAL and Modula-2.
Compiler	A software tool which translates a file from one form to another. For example a Modula-2 Compiler translates a source text file containing a Modula-2 program into machine code.

Modula-2 Software Construction Set Reference Guide

Cursor	A marker which indicates where data typed on the keyboard will be displayed on the screen.
Debug	To analyze and correct errors in a program.
Debugger	A software tool which is used to investigate errors in a program. A debugger allows interactive execution of a user program.
Decimal	A number of base 10. A number represented using the digits between 0 and 9.
Editing	The process of changing, adding or deleting text in a file.
Editor	A program that is used to edit text data interactively. The EMACS program in this package is a good example of an interactive editor.
EMACS	A screen based text editor originally designed at the MIT Artificial Intelligence Laboratory. An EMACS based editor is included with this package.
File	A named storage area used for storing data such as programs and text. Operations may be performed on a file such as copying, deleting, editing the file.
Filename	The name associated with a file storage area. The format for the filename depends on the operating system being used.
Floating Point	A number which consists of an integer part, a fractional part and an exponent. Floating point numbers are used extensively in math calculations.
Function	A self contained piece of code which performs a specific operation. The function typically takes a number of input parameters and returns a single result. The word function is used interchangeably with the word procedure.
Hexadecimal	A number of base 16. A hexadecimal number is represented by a sequence of digits 0 to 9 and the letters A to F.
Identifier	A sequence of characters used in a program to identify an object symbolically.
Initialization	The part of a computer program which is involved in setting up internal information before the main part of the program begins execution.

Intuition	The Amiga user interface software. This software is responsible for maintaining screens, windows, requesters, gadgets and menus.
Linker	A program which binds one or more separate components of a program into a single executable file.
Macro	A piece of data which will be used many times. The data is stored in a temporary place and can be called up by issuing a quick command sequence.
Mark	A marker used in the EMACS editor. The marker can be used for specifying the beginning or end of a region of text. The mark may also be used to move from one end of a buffer to another very quickly.
Memory	A storage area in a computer system. The storage may be used to store programs or data to be manipulated.
Octal	A number represented in base 8. An octal number consists of one or more digits which range from 0 to 7.
Pathname	A direct trail to be followed to gain access to a file. The trail is relative to the current position. The path may be used to access a file not directly available from the current directory.
Point	The location of the cursor in the current EMACS buffer. The text between the point and the mark is called the current region. All text related operations take place from the point.
Prompt	A request for information from the user. The prompt usually consists of a message describing the information needed. The user types in the requested information and then the program continues.
Procedure	A self contained piece of code which performs a specific operation. The procedure may accept input parameters which specify the data to be manipulated or the task to be performed. This term is used interchangeably with the term function.
Recursion	A function which in the course of execution calls itself directly or indirectly. This type of function is useful in breaking down a problem into smaller pieces.

Modula-2 Software Construction Set Reference Guide

- Region** This term as applied to the EMACS editor refers to the text between the point and the mark. EMACS contains many operations which can effect a region of text such as deleting, inserting or copying the region.
- Scope** As applied to identifiers refers to the visibility of an identifier in different sections of a program.
- Source Level Debugger** A debugger which operates on the original high level language source code, rather than at the assembly language level. This type of debugger also allows variables to be examined in the format which corresponds to their real type.
- Stack** A block of memory which is used to store objects temporarily. Objects placed onto the stack must be removed in reverse order from the stack.
- Stream** A high-level virtual data object. A stream is normally connected to a device or file. A stream may be used to input or output data without actually knowing the type of device being accessed.
- Symbolic Debugger** An assembly language debugger capable of displaying symbolic information rather than purely numerical information. The debugger typically display the identifiers used in the original program.
- Window** A rectangle on the video display in which data may be displayed.
- WorkBench** The icon based AmigaDOS operating system user interface. Files may be manipulated from the WorkBench using mouse movements. Programs may also be executed from the WorkBench.

Appendix I Modula-2 Language Syntax.....3

The syntax of the Modula-2 language is described here in Extended Backus-Naur Formalism (EBNF). The rules for EBNF are as follows:

Symbol	Meaning
-----	-----
=	expression equivalent to symbol
	one or the other expression
[]	expression may appear zero or one times
{ }	expression may appear zero or more times
" "	literal symbol in quotes

ident = letter {letter | digit}.

number = integer | real.

integer = digit {digit} ["D"] | octalDigit {octalDigit} ("B"|"C") |
digit {hexDigit} "H".

real = digit {digit} "." {digit} [ScaleFactor].

ScaleFactor = "E" ["+"|" -"] digit {digit}.

hexDigit = digit | "A"|"B"|"C"|"D"|"E"|"F".

digit = octalDigit | "8"|"9".

octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".

string = '"' {character} '"' | ''' {character} '''.

qualident = ident { "." ident }.

ConstantDeclaration = ident "=" ConstExpression.

ConstExpression = expression.

TypeDeclaration = ident "=" type.

type = SimpleType | ArrayType | RecordType | SetType |
PointerType | ProcedureType.

SimpleType = qualident | enumeration | SubrangeType.

enumeration = "(" IdentList ")".

IdentList = ident { "," ident }.

SubrangeType = [qualident] "[" ConstExpression ".." ConstExpression "]".

ArrayType = ARRAY SimpleType { "," SimpleType } OF type.

Modula-2 Software Construction Set Reference Guide

RecordType = RECORD FieldListSequence END.

FieldListSequence = FieldList {";" FieldList}.

FieldList = [IdentList ":" type |
CASE [ident] ":" qualident OF variant {"|" variant}
[ELSE FieldListSequence] END].

variant = [CaseLabelList ":" FieldListSequence].

CaseLabelList = CaseLabels {"," CaseLabels}.

CaseLabels = ConstExpression [".." ConstExpression].

SetType = SET OF SimpleType.

PointerType = POINTER TO type.

ProcedureType = PROCEDURE [FormalTypeList].

FormalTypeList = "(" [[VAR] FormalType
{"," [VAR] FormalType}] ")" [":" qualident].

VariableDeclaration = IdentList ":" type.

designator = qualident {"." ident | "[" ExpList "]" | "^"}.

ExpList = expression {"," expression}.

expression = SimpleExpression [relation SimpleExpression].

relation = "=" | "<" | "<=" | ">" | ">=" | IN.

SimpleExpression = ["+" | "-"] term {AddOperator term}.

AddOperator = "+" | "-" | OR.

term = factor {MulOperator factor}.

MulOperator = "*" | "/" | DIV | REM | MOD | AND | "&" |

factor = number | string | set | designator [ActualParameters] |
"(" expression ")" | NOT factor | "~" factor.

set = [qualident] "{" [element {"," element}] "}".

element = ConstExpression [".." ConstExpression].

ActualParameters = "(" [ExpList] ")".

statement = [assignment | ProcedureCall |
IfStatement | CaseStatement | WhileStatement |

RepeatStatement | LoopStatement | ForStatement |
WithStatement | EXIT | RETURN [expression]].

assignment = designator ":" expression.

ProcedureCall = designator [ActualParameters].

StatementSequence = statement {";" statement}.

IfStatement = IF expression THEN StatementSequence
 {ELSIF expression THEN StatementSequence}
 [ELSE StatementSequence] END.

CaseStatement = CASE expression OF case {"|" case}
 [ELSE StatementSequence] END.

case = [CaseLabelList ":" StatementSequence].

WhileStatement = WHILE expression DO StatementSequence END.

RepeatStatement = REPEAT StatementSequence UNTIL expression.

ForStatement = FOR ident ":" expression TO expression
 [BY ConstExpression] DO StatementSequence END.

LoopStatement = LOOP StatementSequence END.

WithStatement = WITH designator DO StatementSequence END .

ProcedureDeclaration = ProcedureHeading ";" (block ident | FORWARD).

ProcedureHeading = PROCEDURE ident [FormalParameters].

block = {declaration} [BEGIN StatementSequence] END.

declaration = CONST {ConstantDeclaration ";" } |
 TYPE {TypeDeclaration ";" } |
 VAR {VariableDeclaration ";" } |
 ProcedureDeclaration ";" | ModuleDeclaration ";" .

FormalParameters =
 "(" [FPSection {";" FPSection}] ")" [":" qualident].

FPSection = [VAR] IdentList ":" FormalType.

FormalType = [ARRAY OF] qualident.

ModuleDeclaration =
 MODULE ident [priority] ";" {import} [export] block ident.

priority = "[" ConstExpression "]".

Modula-2 Software Construction Set Reference Guide

export = EXPORT [QUALIFIED] IdentList ";"

import = [FROM ident] IMPORT IdentList ";"

DefinitionModule = DEFINITION MODULE ident ";"
 {import} {definition} END ident " ."

definition = CONST {ConstantDeclaration ";" } |
 TYPE {ident ["=" type] ";" } |
 VAR {VariableDeclaration ";" } |
 ProcedureHeading ";" .

ProgramModule = MODULE ident [priority] ";" {import} block ident " ."

CompilationUnit = DefinitionModule | [IMPLEMENTATION] ProgramModule.

Appendix J Bibliography.....	3
------------------------------	---

This appendix contains a list of some of the available Modula-2 books and technical Amiga documentation. There are many other books which are available but may not be listed here.

Modula-2 Introduction

Modula-2: A Seafaring Manual and Shipyard Guide
E. Joyce
Addison-Wesley Publishing Co.

Modula-2 Advanced

Programming in Modula-2, 3rd Edition
Niklaus Wirth
Springer-Verlag

Modula-2 Programming
John W. L. Ogilvie
McGraw-Hill Book Co.

Modula-2 Miscellaneous

Modula-2 for Pascal Programmers
R. Gleaves
Springer-Verlag

Amiga Technical Documents

The AmigaDOS Manual
Commodore-Amiga
Bantam Books Inc.

ROM Kernel Reference Manual: Exec
ROM Kernel Reference Manual: Libraries and Devices
Intuition Reference Manual
Hardware Reference Manual
Commodore-Amiga
Addison-Wesley Publishing Co.

